


Dynamic execution unit management for high performance server system

Patent Number: ☐ [EP0794490](#), [A3](#)
Publication date: 1997-09-10
Inventor(s): YEUNG LEO YUE TAK (US); SHARMA MOHAN (US)
Applicant(s): IBM (US)
Requested Patent: ☐ [JP9265409](#)
Application Number: EP19970300612 19970130
Priority Number(s): US19960613106 19960308
IPC Classification: G06F9/46; H04L29/06
EC Classification: [H04L29/06](#), [G06F9/46A2](#), [G06F9/46R6](#)
Equivalents: JP3229237B2, KR253930, ☐ [US6182109](#)
Cited patent(s): [EP0694837](#); [EP0384339](#); [EP0413490](#); [EP0666665](#); [EP0473913](#)

Abstract

A method, system and product for dynamically managing a pool of execution units in a server system, the pool devoted to a communication process between client and server processes. A minimum and a maximum number of execution units in the communication process pool is established. The minimum number of execution units is the number necessary to support a typical client load. The maximum number of execution units is an upper bound to support a peak client load without overloading the server system. As client requests for service are received by the server system, a number of determinations are made. It is determined whether assigning an execution unit to the request would bring a current number of execution units in the communication process pool over the maximum number of execution units. If so, the client request is rejected. It is determined whether assigning an execution unit to the request would bring the number of assigned execution units to a client task making the request over an allotted number of execution units for the client task. If so, the client request is rejected. The client request if the determinations are negative thereby assigning an execution unit in the communication process pool to the client request. The number of unused execution units in the communication pool is periodically reviewed to determine whether it

should be increased or decreased to improve system performance. 

Data supplied from the esp@cenet database - I2

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平9-265409

(43) 公開日 平成9年(1997)10月7日

(51) Int.Cl. ⁶	識別記号	庁内整理番号	F I	技術表示箇所
G 0 6 F 9/46	3 6 0		G 0 6 F 9/46	3 6 0 B
13/00	3 5 7		13/00	3 5 7 Z

審査請求 未請求 請求項の数 3 O L (全 38 頁)

(21) 出願番号 特願平9-40713

(22) 出願日 平成9年(1997)2月25日

(31) 優先権主張番号 08/613106

(32) 優先日 1996年3月8日

(33) 優先権主張国 米国 (US)

(71) 出願人 390009531

インターナショナル・ビジネス・マシーンズ・コーポレーション

INTERNATIONAL BUSINESS MACHINES CORPORATION

アメリカ合衆国10504、ニューヨーク州
アーモンク (番地なし)

(72) 発明者 モーハン・シャルマ

アメリカ合衆国78728 テキサス州オース
チン ルネッサンス・コート 14000 ナ
ンバー1102

(74) 代理人 弁理士 合田 潔 (外2名)

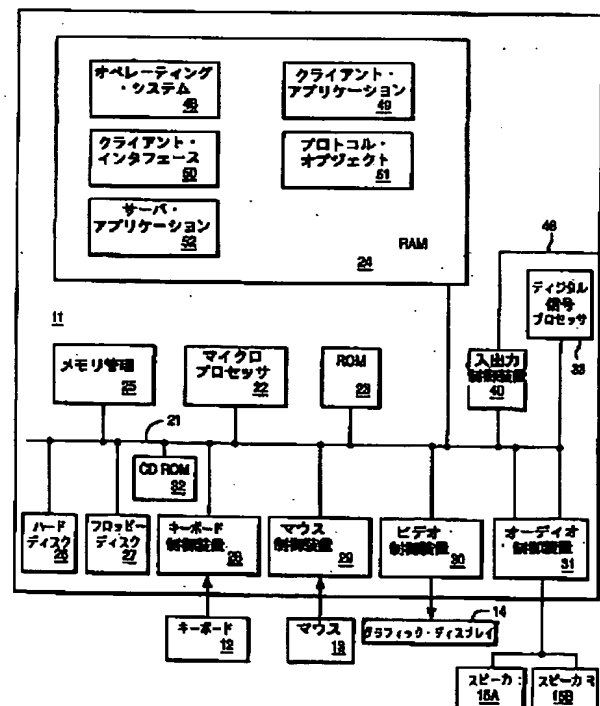
最終頁に続く

(54) 【発明の名称】 高性能ユーザ・レベル・ネットワーク・プロトコル・サーバ・システムのための動的実行ユニット管理

(57) 【要約】

【課題】 サーバ・システム内の実行ユニットのプールを動的に管理するための方法、システム、製品を提供する。

【解決手段】 そのプールは、クライアント・プロセスとサーバ・プロセスとの間の通信プロセス専用である。通信プロセス・プール内に最小数および最大数の実行ユニットが確立される。最小数の実行ユニットは、典型的なクライアント・ロードをサポートするのに必要な数である。最大数の実行ユニットは、サーバ・システムにとって過負荷にならずにピーク・クライアント・ロードをサポートするための上限である。クライアントによるサービス要求がサーバ・システムによって受信されると、いくつかの判定が行われる。まず、その要求に実行ユニットを割り当てると、通信プロセス・プール内の現行数の実行ユニットが最大数の実行ユニットを上回るようになるかどうか判定される。上回る場合、クライアント要求は拒否される。



【特許請求の範囲】

【請求項1】サーバ・システム内の実行ユニットの、クライアント・プロセスとサーバ・プロセスとの間の通信プロセス専用のプールを動的に管理するための方法において、

通信プロセス・プール内に、典型的なクライアント・ロードをサポートするのに必要な数である最小数、およびサーバ・システムにとって過負荷にならずにピーク・クライアント・ロードをサポートするための上限である最大数の実行ユニットを割り振るステップと、

クライアントによるサービス要求をサーバ・システムが受信するステップとを含み、

受信した各クライアント要求ごとに、

受信したクライアント要求に実行ユニットを割り当てると、通信プロセス・プール内の現行数の実行ユニットが最大数の実行ユニットを上回ることになるかどうかを判定し、上回る場合にクライアント要求を拒否するステップと、

受信したクライアント要求に実行ユニットを割り当てると、その要求を行うクライアント・タスクに割り当てられた現行数の実行ユニットがクライアント・タスク用の割り当てられた実行ユニットの数を上回ることになるかどうかを判定し、上回る場合にクライアント要求を拒否するステップと、

判定ステップが否定であればクライアント要求を許可し、その結果、通信プロセス・プール内の実行ユニットがクライアント要求に割り当てられるステップとを含むことを特徴とする方法。

【請求項2】サーバ・システム内の実行ユニットの、クライアント・プロセスとサーバ・プロセスとの間の通信プロセス専用のプールを動的に管理するためのシステムにおいて、

通信プロセス・プール内に、典型的なクライアント・ロードをサポートするのに必要な数である最小数、およびサーバ・システムにとって過負荷にならずにピーク・クライアント・ロードをサポートするための上限である最大数の実行ユニットを割り振る手段と、

クライアントによるサービス要求をサーバ・システムが受信する手段と、

受信したクライアント要求に実行ユニットを割り当てると、通信プロセス・プール内の現行数の実行ユニットが最大数の実行ユニットを上回ることになるかどうかを判定する手段と、

受信したクライアント要求に実行ユニットを割り当てると、その要求を行うクライアント・タスクに割り当てられた現行数の実行ユニットがクライアント・タスク用の割り当てられた実行ユニットの数を上回ることになるかどうかを判定する手段と、

通信プロセス・プール内の実行ユニットをクライアント要求に割り当てることができると判定手段が確認した場

合にクライアント要求を許可する手段とを含むことを特徴とするシステム。

【請求項3】サーバ・システム内の実行ユニットの、クライアント・プロセスとサーバ・プロセスとの間の通信プロセス専用のプールを動的に管理するためのコンピュータが読取り可能な媒体上のコンピュータ・プログラム製品において、

通信プロセス・プール内に、典型的なクライアント・ロードをサポートするのに必要な数である最小数、およびサーバ・システムにとって過負荷にならずにピーク・クライアント・ロードをサポートするための上限である最大数の実行ユニットを割り振るようにシステムに指示する手段と、

クライアントによるサービス要求をサーバ・システムが受信するようにシステムに指示する手段と、

受信したクライアント要求に実行ユニットを割り当てると、通信プロセス・プール内の現行数の実行ユニットが最大数の実行ユニットを上回ることになるかどうかを判定するようにシステムに指示する手段と、

受信したクライアント要求に実行ユニットを割り当てると、その要求を行うクライアント・タスクに割り当てられた現行数の実行ユニットがクライアント・タスク用の割り当てられた実行ユニットの数を上回ることになるかどうかを判定するようにシステムに指示する手段と、

通信プロセス・プール内の実行ユニットをクライアント要求に割り当てることができるという判定にตอบสนองしてクライアント要求を許可するようにシステムに指示する手段とを含むことを特徴とするコンピュータ・プログラム製品。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、一般に、コンピュータ・ネットワーク上の通信、ならびにこのような通信を容易にするコンピュータ・プロトコルに関する。より具体的には、本発明は、ネットワーク・プロトコル・アクセスのためのオブジェクト指向通信インタフェースに関する。

【0002】

【従来の技術】非常に初期のコンピュータ・システムは、ディスプレイやプリンタなどの周辺装置と入力装置が接続された、スタンドアロン・プロセッサであった。各コンピュータ・システムは独立しており、コンピュータ・システム間の通信はほとんど行われていなかった。現在、ローカル・エリア・ネットワークや広域ネットワークなどのコンピュータ・ネットワークでは、複数のコンピュータ・システムを相互接続し、ネットワークに結合された各種コンピュータ・システムから得られるデータ、サービス、資源の共有を含む様々な利益を達成することが周知になっている。

【0003】ネットワークにより各種コンピュータ・シ

3

システム間で通信するために、多くの通信プロトコルが開発された。周知のネットワーク・プロトコルの例としては、システム・ネットワーク体系（SNA）、伝送制御プロトコル／インターネット・プロトコル（TCP／IP）、ネットワーク基本入出力システム（Net BIOS）、インターネット・パケット交換／順次パケット交換（IPX／SPX）などがある。その他の通信プロトコルも既知であり、広く使用され、ISO、IEEE、その他の組織の様々な規格に記載されている。コンピュータ・ネットワークの理解を容易にするため、ネットワークの諸機能および関連ソフトウェアは一連の層として記述されることが多い。ネットワークにより配布されたアプリケーションの1つのコピーとその配布されたアプリケーションの別のコピーとの間のデータ転送は、基礎となる一連の通信層のサービスを使用することによって行われる。一般に、1つのコンピュータ・システム内の各層は、それぞれの層がそれぞれの対等層とやりとりするように、受信側コンピュータ・システム内に対応層を備えている。

【0004】7層の開放型システム間相互接続（OSI）モデルは、ネットワーク通信の説明のうち、最もよく知られたものの1つであるが、多くの通信実施態様では、1つまたは複数のOSI層を結合するかまたは省略している。OSIの物理層は、ネットワークと直接やりとりする最下層である。これは、物理的接続部を越えて実際にビット・ストリームをネットワークに伝送することを含む。第2の層は、物理層ストリームの多重化とフレーム化を行ってメッセージにするデータリンク層である。これは、エラー検出、同期情報、物理チャネル管理ももたらす。第3の層は、ネットワークによる情報の経路指定を制御するネットワーク層である。アドレス指定、ネットワークの初期設定、切替え、セグメント化、フォーマット化などのサービスはこの層に用意される。データ送達の肯定応答は、この層で行われる場合もあれば、データリンク層で行われる場合もある。

【0005】第4の層は、透過データの送達、多重化、マッピングを制御するトランスポート層である。これより下の諸層で行われる最善の努力とは対照的に、アプリケーションが必要とすれば、信頼性の高い送達がこの層で行われる。欠落データの再伝送、順不同で送達されたデータの再配列、伝送エラーの訂正などのサービスは、通常、この層で行われる。第5の層は、トランスポート層からの情報を使用して、セッションと呼ばれるネットワーク内の2つのノード間の共通活動として個々のデータをグループ化するセッション層である。第6の層は、セッション層と第7の層であるアプリケーション層とのインタフェースを含むプレゼンテーション層である。プレゼンテーション層は、セッション層の整合性を損なわずに、アプリケーション層で使用するための情報を提供する。また、プレゼンテーション層はデータの解釈とフ

4

ォーマットおよびコードの変換を行うのに対し、アプリケーション層はユーザ・アプリケーション・インタフェースと管理機能を提供する。

【0006】もう1つの周知のネットワーク規格はIEEE規格である。IEEEモデルとOSIとの主な相違点は、OSIの第2の層であるデータリンク層が媒体アクセス制御（MAC）副層と論理リンク制御（LCC）副層という2つの副層に分割されることである。媒体アクセス制御は、その制御アクセスにおける通信媒体への媒体アクセス接続を管理する。論理リンク制御は、関連するデータ・リンク制御が指定するプロトコルをサポートするための状態マシンを提供する。

【0007】もう1つの周知の技術は、オブジェクトというプログラミング・エンティティにデータとメソッドをカプセル化するオブジェクト指向プログラミングである。公用インタフェースにより所与のメソッドとデータを保護することにより、オブジェクト指向プログラムは、各コンポーネントを他のコンポーネントに対する変更から隔離しながら、最小限の再プログラミングによって必要な諸機能を提供することができる。オブジェクト指向の技術、概念、規則に関する詳細な背景情報については、グレイディー（Grady Booch）によるObject Oriented Design With Applications (The Benjamin/Cummings Publishing Company, 1990)およびB.マイヤー（Meyer）によるObject Oriented Software Construction (Prentice Hall, 1988)などの参考文献を参照されたい。

【0008】マルチプロセッサ・ネットワークにおける通信プロトコルの一般領域に対してオブジェクト指向技術を応用しようという試みは、すでにいくつか行われてきた。以下に示すように、それは依然として本発明の実り多い分野である。

【0009】

【発明が解決しようとする課題】サーバ・システム内の実行ユニットのプールを動的に管理するための方法、システム、製品であって、そのプールは、クライアント・プロセスとサーバ・プロセスとの間の通信プロセス専用である。通信プロセス・プール内に最小数および最大数の実行ユニットが確立される。最小数の実行ユニットは、典型的なクライアント・ロードをサポートするのに必要な数である。最大数の実行ユニットは、サーバ・システムにとって過負荷にならずにピーク・クライアント・ロードをサポートするための上限である。クライアントによるサービス要求がサーバ・システムによって受信されると、いくつかの判定が行われる。まず、その要求に実行ユニットを割り当てると、通信プロセス・プール内の現行数の実行ユニットが最大数の実行ユニットを上回ることになるかどうか判定される。上回る場合、クライアント要求は拒否される。次に、その要求に実行ユニットを割り当てると、その要求を行うクライアント・タスクに割り当てられた実行ユニットの数がクライアン

5

ト・タスク用の割り当てられた実行ユニットの数を上回ることになるかどうか判定される。上回る場合、クライアント要求は拒否される。判定結果が否定であればクライアント要求が許可され、それにより、通信プロセス・プール内の実行ユニットがクライアント要求に割り当てられる。

【0010】

【課題を解決するための手段】システム・パフォーマンスを改善するために通信プール内の未使用の実行ユニットの数を増減する必要があるかどうかを判定するため、未使用の実行ユニットの数を定期的に検討する。

【0011】上記の目的、特徴、および利点は、添付図面と以下の説明を参照すれば、さらに容易に理解されるだろう。

【0012】

【発明の実施の形態】本発明は、複数通りのオペレーティング・システム下で様々なコンピュータまたは複数のコンピュータの集合で動作することができる。このコンピュータは、たとえば、パーソナル・コンピュータ、ミニ・コンピュータ、メインフレーム・コンピュータ、他のコンピュータからなる分散ネットワークで動作するコンピュータなどにすることができる。コンピュータの具体的な選択はディスクやディスク記憶装置の要件によってのみ制限されるが、IBMのPCシリーズのコンピュータであれば、本発明に使用できるはずである。IBMのPCシリーズのコンピュータの詳細については、IBM PC 300/700 Series Hardware Maintenance, Publication No. S83G-7789-03およびUser's Handbook IBM PC Series 300 and 700, Publication No. S83G-9822-00等を参照されたい。IBMのパーソナル・コンピュータが動作可能なオペレーティング・システムの1つは、IBMのOS/2 Warp 3.0である。IBMのOS/2 Warp 3.0オペレーティング・システムの詳細については、OS/2 Warp V3 Technical Library, Publication No. GBOF-7116-00等を参照されたい。

【0013】代替実施例のコンピュータ・システムは、AIX (TM) オペレーティング・システム上で動作するIBMのRISCシステム/6000 (TM) 系列のコンピュータにすることもできる。RISCシステム/6000の様々なモデルについては、RISC System/6000, 7073 and 7016 POWERstation and POWERserver Hardware Technicalという参考文献 (資料番号SA23-2644-00) など、IBMの多くの資料に記載されている。AIXオペレーティング・システムについては、General Concepts and Procedure -- AIX for RISC System/6000

(資料番号SC23-2202-02) ならびにIBMのその他の資料に記載されている。

【0014】図1には、システム・ユニット11と、キーボード12と、マウス13と、ディスプレイ14とを含むコンピュータ10がブロック図形式で示されてい

6

る。システム・ユニット11は、1つのシステム・バスまたは複数のシステム・バス21を含み、このシステム・バスには様々な構成要素が結合され、このシステム・バスにより様々な構成要素間の通信が行われる。マイクロプロセッサ22は、システム・バス21に接続され、同じくシステム・バス21に接続された読取り専用メモリ (ROM) 23とランダム・アクセス・メモリ (RAM) 24によってサポートされている。IBMのPS/2シリーズのコンピュータに搭載されているマイクロプロセッサは、386または486マイクロプロセッサを含む、インテルのマイクロプロセッサ・ファミリーの1つである。しかし、68000、68020、68030マイクロプロセッサなどのモトローラのマイクロプロセッサ・ファミリーや、IBM製のPowerPCチップまたはヒューレット・パッカード、サン、モトローラなどが製造するものなどの様々な縮小命令セット・コンピュータ (RISC) マイクロプロセッサを含み、かつこれらに限定されないマイクロプロセッサも、特定のコンピュータで使用することができる。

【0015】ROM23は、数あるコードのうち、やりとりや、ディスク・ドライブおよびキーボードなどの基本的なハードウェア操作を制御する、基本入出力システム (BIOS) を含んでいる。RAM24は、オペレーティング・システムとアプリケーション・プログラムのロード先になるメイン・メモリである。メモリ管理チップ25は、システム・バス21に接続され、RAM24とハード・ディスク・ドライブ26およびフロッピー・ディスク・ドライブ27とのデータの受渡しを含む直接メモリ・アクセス操作を制御する。同じくシステム・バス21に結合されたCD-ROM32は、マルチメディア・プログラムまたはプレゼンテーションなどの大量のデータを格納するために使用する。

【0016】このシステム・バス21には、様々な入出力制御装置、すなわち、キーボード制御装置28、マウス制御装置29、ビデオ制御装置30、オーディオ制御装置31も接続されている。予想通り、キーボード制御装置28はキーボード12用のハードウェア・インタフェースになり、マウス制御装置29はマウス13用のハードウェア・インタフェースになり、ビデオ制御装置30はディスプレイ14用のハードウェア・インタフェースであり、オーディオ制御装置31はスピーカ15用のハードウェア・インタフェースである。トークン・リング・アダプタなどの入出力制御装置40により、ネットワーク46を介して同様に構成された他のデータ処理システムへの通信が可能になる。

【0017】本発明の好ましい実施態様の1つは、上記のように一般的に構成された1つまたは複数のコンピュータ・システムのランダム・アクセス・メモリ24に常駐する複数の命令セット48~52である。コンピュータ・システムが要求するまで、命令セットは、ハード・

ディスク・ドライブ26などの他のコンピュータ・メモリ、最終的にCD-ROM32で使用するための光ディスクなどの取外し可能メモリ、最終的にフロッピー・ディスク・ドライブ27で使用するためのフロッピー・ディスクに格納することができる。当業者であれば、この命令セットを物理的に格納すると、それが電氣的、磁氣的、または化学的に格納されている媒体が物理的に変化し、その媒体上にコンピュータが読取り可能な情報が乗ることを理解できるはずである。命令、記号、文字などによって本発明を記述すると便利であるが、これらの表現および同様の表現はいずれも適切な物理要素に関連付けられている必要があることに留意されたい。さらに、本発明は、比較や妥当性検査、あるいは人間の操作員に関連しう他の表現によって記述されることも多い。ここに記載するいずれの操作でも、本発明の一部を形成するような人間の操作員によるアクションは一切不要であり、この操作は電気信号を処理して他の電気信号を生成するためのマシン操作である。

【0018】このワークステーションが統合されているネットワークは、ローカル・エリア・ネットワーク(LAN)または広域ネットワーク(WAN)であり、後者は他のノードまたは既知のコンピュータ・アーキテクチャに基づいて動作するシステムのネットワークへのテレプロセッシング接続を含む。いずれのノードでも、1つまたは複数の処理システムが存在可能であり、それぞれの処理システムはだいたい上記のように構成された単一ユーザ・システムまたは複数ユーザ・システムにすることができる。これらの処理システムは、サービスを要求するか供給するかに応じて、クライアント・ワークステーションまたはサーバ・ワークステーションとして動作する。特定の実施態様では、本発明は、様々な通信プロトコルのうちの1つまたは複数によって通信するネットワークによって相互接続された複数のIBM互換ワークステーション上で動作する。ソフトウェア・アプリケーションは、まとめてパッケージ化するか、個別のアプリケーションとして販売することができる。ローカル・エリア・ネットワークの簡単な説明は、ラリーE. ジョーダン(Larry E. Jordan)およびブルース・チャーチル(Bruce Churchill)著Communications and Networking For The IBM PC(Robert J. Brady発行、A Prentice Hall Company 1983)等に記載されている。

【0019】好ましい実施例では、本発明は、オブジェクト指向プログラミング技法を使用してC++プログラミング言語で実現される。C++はコンパイル済みの言語である。プログラムは人間が読取り可能なスクリプトで作成され、このスクリプトがコンパイラという他のプログラムに供給され、マシンが読取り可能な数字コードが生成されるが、このコードはコンピュータにロードしてコンピュータが直接実行できるものである。C++言語は、ソフトウェア開発者が他の開発者によって作成さ

れたプログラムを容易に使用できるようにするような参与の特性を処理しながら、その破壊や不適正使用を防止するためにプログラムの再使用を大幅に制御する。C++言語は周知のものであるので、この言語について詳細に記載した論文やテキストは数多く存在する。

【0020】当業者には既知のように、オブジェクト指向プログラミング技法は、「オブジェクト」の定義、作成、使用、破棄を含む。これらのオブジェクトは、データ要素とルーチン、すなわちデータ要素を操作するメソッドとを含む、ソフトウェア・エンティティである。データと関連メソッドは、ソフトウェアによってエンティティとして扱われ、そのように作成し、使用し、削除することができる。データと関数により、オブジェクトは、データ要素により提示可能なその属性と、そのメソッドにより表現可能なその挙動によって、実世界のエンティティをモデル化することができる。

【0021】オブジェクトの定義は、オブジェクトそのものではないが、実際のオブジェクトを構築する方法をコンパイラに指示するテンプレートとして機能する「クラス」を作成することによって行う。たとえば、クラスは、データを操作する関数に含まれるデータ変数とステップの数とタイプを指定することができる。実際にはオブジェクトは、対応するクラス定義と、オブジェクト作成中に供給される引数などの追加情報とを使用してオブジェクトを構築する、コンストラクタという特殊な関数を使ってプログラム内に作成される。オブジェクトの破棄は、デストラクタという特殊な関数によって行われる。

【0022】多くの利点は、オブジェクト指向プログラミング技法の3つの基本的特性、すなわち、カプセル化、多様性、継承から発生している。オブジェクトは、内部データ構造および内部関数の全部または一部を隠す、すなわち、カプセル化するために設計することができる。より具体的には、プログラム設計時にプログラム開発者は、データ変数の全部または一部と関連メソッドの全部または一部を「私用」すなわちそのオブジェクトのみが使用するものと見なしてオブジェクトを定義することができる。他のデータまたはメソッドは、「公用」すなわち他のプログラムが使用可能なものであると宣言することができる。他のプログラムによる私用変数およびメソッドへのアクセスは、そのオブジェクトの私用データにアクセスする公用メソッドを定義することによって制御することができる。この公用メソッドは、私用データと外部プログラムとのインタフェースを形成する。私用変数に直接アクセスするプログラム・コードを作成しようと試みると、コンパイラは、プログラムのコンパイル時にエラーを発生する。このエラーによって、コンパイル・プロセスが停止し、プログラムが実行できなくなる。

【0023】多様性とは、全体的なフォーマットが同じ

であるが別々のデータで機能する複数のオブジェクトおよび関数が、別々に機能して一定の結果を生み出せるようにするものである。たとえば、加算メソッドを変数Aプラス変数B (A+B) として定義することができる。AとBが数字、文字、またはドルとセントであっても、これと同じフォーマットを使用することができる。しかし、この加算を実行する実際のプログラム・コードは、AとBを構成する変数のタイプに応じて、大幅に異なってくる可能性がある。したがって、変数のタイプ(数字、文字、ドル)ごとに1つずつ、3通りのメソッド定義を作成することができる。メソッドの定義後、プログラムはその共通フォーマット (A+B) によってその加算メソッドをあとで参照することができ、コンパイル時にC++コンパイラは、その変数タイプを検査することによって3つのメソッドのいずれを使用すべきかを判定する。その後、コンパイラは適切な関数コードを代入する。

【0024】オブジェクト指向プログラミングの第3の特性は、プログラム開発者が既存のプログラムを再使用できるようにする継承である。継承により、ソフトウェア開発者は、クラスと、クラス階層によって関連付けられるようにそのクラスからあとで作成されるオブジェクトとを定義することができる。具体的には、クラスは、他の基本クラスのサブクラスと呼ぶこともできる。サブクラスは、その基本クラスの公用関数がサブクラスに含まれている場合のように、その公用関数のすべてを「継承」し、そのすべてにアクセスすることができる。また、サブクラスは、同じ形式の新しい関数を定義することによって、継承した関数の一部または全部を指定変更したり、修正することもできる。

【0025】他のクラスの機能性を借用する新しいサブクラスを作成すると、ソフトウェア開発者は、その特定のニーズに合うように既存のコードを容易にカスタマイズすることができる。

【0026】オブジェクト指向プログラミングにより、他のプログラミングの概念が大幅に改善されるが、特に修正に使用可能な既存のソフトウェア・プログラムがまったくない場合には、依然としてプログラム開発には相当な時間と労力が必要になる。その結果、いずれも特定の環境で一般に検出されるタスクの実行向けの1組のオブジェクトと追加の雑ルーチンを作成するために、1組の事前定義相互接続済みクラスが用意されていることがある。このような事前定義クラスとライブラリは、通常、「フレームワーク」と呼ばれ、本質的に作業アプリケーション用の事前製作構造を提供する。

【0027】たとえば、ユーザ・インタフェース用のフレームワークは、ウィンドウ、スクロール・バー、メニューなどを作成する1組の事前定義グラフィック・インタフェース・オブジェクトを提供し、これらのグラフィック・インタフェース・オブジェクト用のサポートと

「デフォルト」挙動を提供する可能性がある。多くのフレームワークはオブジェクト指向技法に基づいているので、事前定義クラスを基本クラスとして使用することができ、組込みデフォルト挙動を開発者定義サブクラスが継承し、開発者がそのフレームワークを拡張し、特定の専門分野でカスタマイズ済みの解決策を作成できるように修正または指定変更することができる。プログラマは元のプログラムを変更せず、むしろ元のプログラムの諸機能を拡張するので、このオブジェクト指向手法は従来のプログラミングより大幅に優れている。さらに、このフレームワークは体系的なガイダンスとモデル化を提供し、同時に、開発者は問題定義域に固有の特定のアクションを自由に供給できるようになる。

【0028】以下に記載する本発明では、コンピュータ・ネットワーク内の様々なコンピュータ・システムのアプリケーション用のネットワーク・サービスを提供するためにネットワーキング・フレームワークを提供する。

【0029】プロトコル・インタフェース・モデル

ここでは、オブジェクト指向のプロトコル・インタフェース・モデルおよび機構について説明する。このインタフェースの一般的な特徴により、OSIネットワーク・モデルのすべての層の開発が可能になる。したがって、すべてのネットワーク層は同様の構文になり、その意味は特定の層の仕様によって定義される。すなわち、OSIネットワーク・モデル内の各種の層を実現するオブジェクトは構文上は同様であるが、そのインプリメンテーションは特定の層の責任に応じて異なる可能性がある。さらに、TCP/IPなどの特定のプロトコル用のそれぞれの層を実現するオブジェクトは、NetBIOSなどの別のプロトコルの同様の層とは、機能およびインプリメンテーションの点で異なる可能性がある。というのは、2つのプロトコルではそれぞれの層の責任が異なるからである。このモデルは、通信端点を定義し、端点を再使用し、ネットワーク事象を監視するための機構も提供する。これは、オブジェクト指向モデルなので、コードの再使用性、保守容易性、クライアント・アプリケーションに影響せずにインプリメンテーションを更新する能力など、オブジェクト指向インプリメンテーションの特徴をすべて継承している。

【0030】1. 通信端点の作成：通信端点の作成は、ネットワーク定義オブジェクトにより容易になっている。各ネットワーク定義オブジェクトは、クライアント・インタフェースの定義を含む。これは、クライアント・プログラムがアクセスを必要とすると思われる各種タイプのオブジェクトの抽象概念である。通信端点は、ネットワーク定義クラスから派生したTAccessDefinitionオブジェクトである。

【0031】図2は、ネットワーク定義オブジェクトのクラス階層を示している。TNetworkDefinitionクラス100は、端点をインスタンス化するためのメソッドInst

antiateDefinitionと、通信端点を破棄するためのメソッドDeinstantiationDefinitionをそれぞれ含んでいる。TNetworkDefinitionの新しいインスタンスを構築するためのメソッドまたは特定のクラス・オブジェクトを破棄するためのメソッドも用意されているが、これらのメソッドはすべてのクラスに共通なので、以下の説明では繰り返さないことにする。

【0032】TAccessDefinitionオブジェクト101は、特定のプロトコル・タイプとその諸層を定義するプロトコル・インタフェース・オブジェクトを含んでいる。TAccessDefinitionオブジェクト101は、通信端点のハンドルとして機能する。さらに、TAccessDefinitionオブジェクト101は、その親であるTNetworkDefinitionから継承したメソッドに加え、すでに追加されているものの上にアクセス定義へのインタフェース層を加えるためのメソッドAddToTopと、一番下にすでに追加されているものにAccessDefinitionへのインタフェース層を加えるためのメソッドAddToBottomと、最も高いプロトコル層インタフェースを獲得するためのメソッドGetTopOfStackも含んでいる。TEventDefinitionクラス103は、ネットワーク事象を監視するために使用する。TEventDefinitionクラス103については、以下のNetworkEventの項でより詳細に説明する。

【0033】2. ネットワーク・アドレス・クラス
TNetworkAddressクラス111は、すべてのプロトコル・アドレス・クラスとハードウェア・アドレス・クラスを定義するために使用する基本クラスである。図3は、プロトコル・アドレス・クラスのクラス階層を示している。TNetworkAddressクラス111は、アドレスのタイプをテストするためのメソッドIsOfAddressTypeと、グループ・アドレスかどうかをテストするためのメソッドIsGroupと、同報アドレスかどうかをテストするためのメソッドIsBroadCastと、マルチキャスト・アドレスかどうかをテストするためのメソッドIsMulticastと、ヌル・アドレスかどうかをテストするためのメソッドIsNullAddressとを含んでいる。このクラスは、アドレスのワイヤ長を獲得するためのメソッドGetWireLengthと、アドレスをヘッダにフォーマット化するためのメソッドAddressToWireと、ヘッダからアドレスを獲得するためのメソッドWireToAddressも含んでいる。ストリームが端点に着信するのか、それともストリームが端点から発信するのかを判定するための演算子も、このクラスの一部である。各クラスに固有のポインタを獲得するためのメソッドGetClassIdentifierは、このクラスによって提供される。

【0034】TProtocolAddressクラス・オブジェクトのインスタンスは、プロトコル・アドレスを渡すことができるように、通信端点として機能する。TProtocolAddressオブジェクト113は、同報アドレスを獲得するためのメソッドBroadCastAddressと、ヌル・アドレスを獲得

するためのメソッドNullAddressとを加える。THardwareAddressオブジェクト115のインスタンスは、必要に応じてアプリケーションにハードウェア・アドレスを渡すものである。同様に、THardwareAddressオブジェクト115は、同報アドレスを獲得するためのメソッドBroadCastAddressと、ヌル・アドレスを獲得するためのメソッドNullAddressとを加える。また、機能アドレスかどうかをテストするためのメソッドIsFunctionalも加える。TIEEE8023Addressクラスは、IEEE802.3というアドレス指定規則に応じてハードウェア・クラス・アドレスを変更したものである。さらに、これは、グループ・アドレスかどうかをテストするためのメソッドIsGroupと、グループ・アドレスに設定するためのメソッドSetGroupと、グループ・アドレスをリセットするためのメソッドClearGroupとを加える。他のメソッドとしては、同報アドレスを獲得するBroadCastAddress、マルチキャスト・アドレスかどうかをテストするIsMulticast、ヌル・アドレスを獲得するNullAddress、標準入力からアドレスを獲得するCanonicalAddressなどがある。

【0035】TTCPAddr117とTNBAddr119は、それぞれTCP/IPおよびNetBIOSのアドレス指定の詳細を表す具象アドレス・クラスの例である。同様に、TIEEE8023Addr121とTIEEE8025Addr123は、IEEE802.3およびIEEE802.5のアドレス指定用の具象ハードウェア・アドレスを表している。

【0036】3. プロトコル・インタフェース・クラス
本発明は、接続指向のトランザクションと無接続のトランザクションの両方をサポートする。また、本発明では、プロトコルとは無関係のアプリケーションが従う状態マシンについても説明する。プロトコル・インタフェース・クラスのオブジェクト階層を図4に示す。プロトコル・インタフェース・クラスは、プロトコルが備えていなければならないすべての共通関数用のメソッドを含む、MProtocolServe133という基本クラスから派生したものである。TProtocolInterfaceクラス135は、ネットワーク機能用の追加メソッドを含んでいる。これらのメソッドについては、以下の表1と表2に詳しく示す。TCP/IPなどのネットワーク・プロトコルは、そのTCPIPInterfaceクラスをTProtocolInterfaceから派生させて、デフォルト・インプリメンテーションを指定変更し、送信または受信要求上に有効フラグがあるかどうかという検査などのそれ専用の特定の特徴を追加する。TProtocolInterfaceクラスから派生する個別のクラスとしては、セッション層用のTSessionInterface137、トランスポート層用のTTransportInterface138、ネットワーク層用のTNetworkInterface141、ファミリー層用のTFamilyInterface143、データ・リンク層用のTDLInterface145が用意されている。オブジェクト階層は図示の通りである。この実施例では、OSIネットワーク層が、ネットワーク層と、プロトコル

・ファミリー層という下位層とに分割される。ファミリー層は、経路指定情報など、OS Iネットワーク層の複製部分を含んでいる。ネットワーク層は、対等アドレスおよびローカルSAPなど、特定の端点に関連する情報を含んでいる。これは、FamilyLayerオブジェクトなど、1つのオブジェクトだけがシステム内に常駐するようにして、すべてのグローバル情報とプロトコルとの関連を維持するために行われるものである。クライアント・アプリケーションによって各端点が作成され破棄されると、セッション層などの他のプロトコル層オブジェクトと、トランスポート層オブジェクトと、ネットワーク層オブジェクトは作成され破棄されるが、ファミリー層オブジェクトとデータリンク層オブジェクトは常駐したままになる。

Bind	ーローカル・アドレスを初期設定しバインドする
Unbind	ーローカル・アドレスをアンバインドする
SendRelease	ー正常解放開始
ReceiveRelease	ー正常解放開始の肯定応答受信
GetLocalAddress	ーローカル・アドレスを獲得する
SetLocalAddress	ーローカル・アドレスを設定する
GetPeerAddress	ー対等アドレスを獲得する
SetPeerAddress	ー対等アドレスを設定する
GetProtocolInfo	ープロトコル情報を獲得する
SetProtocolInfo	ープロトコル情報を設定する
GetProtocolOptions	ープロトコル・オプションを設定する
GetRequestMode	ー要求モードを獲得する
SetRequestMode	ー要求モードを設定する
GetRetry	ープロトコル層再試行パラメータを獲得する
SetRetry	ープロトコル層再試行パラメータを設定する
GetTimeout	ープロトコル層タイムアウト・パラメータを獲得する
SetTimeout	ープロトコル層タイムアウト・パラメータを設定する
GetStatistics	ープロトコル層統計情報を獲得する
SetStatistics	ープロトコル層統計情報を設定する
IsSession	ープロトコル層がセッション層である場合に真を返す
IsTransport	ープロトコル層がトランスポート層である場合に真を返す
IsNetwork	ープロトコル層がネットワーク層である場合に真を返す
IsFamily	ープロトコル層がファミリー層である場合に真を返す
演算子<<=	ーオブジェクトをデータ・ストリームに受信するための演算子
演算子>>=	ーオブジェクトをデータ・ストリームに送信するための演算子

【0040】表2

TProtocolInterfaceクラスに設けられた関数のリストを

【0037】具象クラスは、個々のインタフェース・クラスから派生する。たとえば、以下に説明するように、プロトコル・スタックを構築するためにTCP/IPなどの特定のプロトコル用のトランスポート、ネットワーク、ファミリー・インタフェース・クラスのために、具象クラスが用意されている。

【0038】表1

前述のように、MProtocolServiceオブジェクトは、プロトコル層定義用の基本クラスとして機能する。MProtocolServiceオブジェクトに設けられているメソッドのリストを以下に示す。これらのメソッドの大部分は、純粹仮想関数である。

【0039】

以下に示す。

【0041】

15	16
GetLayerIndex	—プロトコル層のインデックスを取得する
CancelRequests	—現行未解決要求をすべて取り消す
ReceiveEvent	—このスタック上の受信事象
GetConnectionInfo	—接続情報およびメモリの制約を入手する
BorrowMemory	—ネットワーク・データ用にシステム・メモリを借用する
ReturnMemory	—ネットワーク・データ用のシステム・メモリを返す
GetAccessDefinition	—TAccessDefinitionへのポインタを取得する
GetLocalAddress	—層のローカル・アドレスを取得する
SetLocalAddress	—層のローカル・アドレスを設定する
GetPeerAddress	—層の対等アドレスを取得する
SetPeerAddress	—層の対等アドレスを設定する
GetProtocolInfo	—層のプロトコル情報を取得する
SetProtocolInfo	—層のプロトコル情報を設定する
GetProtocolOptions	—層のプロトコル・オプションを取得する
SetProtocolOptions	—層のプロトコル・オプションを設定する
GetRequestMode	—層の要求モードを取得する
SetRequestMode	—層の要求モードを設定する
GetRetry	—プロトコル層再試行パラメータを取得する
SetRetry	—プロトコル層再試行パラメータを設定する
GetStatistics	—プロトコル層統計情報を取得する
GetTimeout	—プロトコル層タイムアウト・パラメータを取得する
SetTimeout	—プロトコル層タイムアウト・パラメータを設定する
Bind	—プロトコル・スタックをバインドする
Unbind	—プロトコル・スタックをアンバインドする
Receive	—ネットワーク・データを受信する
Send	—ネットワーク・データを送信する
Connect	—接続を確立しようという試みを開始する
ReceiveConnection	—接続を確立しようという試みを待つ
Disconnect	—接続を終了する
ReceiveDisconnect	—切断を待つ
AcceptConnection	—接続を確立しようという試みを受け入れる
RejectConnection	—接続を確立しようという試みを拒否する
Listen	—接続を確立しようという試みを聴取する
ListenForConnection	—接続を確立しようという試みを聴取する
SendRelease	—正常解放開始⇒送信すべきデータはこれ以上ない
ReceiveRelease	—正常解放開始の肯定応答受信
演算子<<=	—オブジェクトをデータ・ストリームにストリーム・インするための演算子
演算子>>=	—オブジェクトをデータ・ストリームにストリーム・アウトするための演算子

【0042】4. プロトコル層インプリメンテーション・クラス

前述のように、プロトコル・インタフェース・モデルは、プロトコル層にアクセスするためのオブジェクト・ベースのAPIとして機能する。TCP/IPなどのプ

ロトコル・スタックのインプリメンテーションを階層化した1組のリンク済みオブジェクトとして実現するには、TProtocolLayerクラスを使用する。図5は、プロトコル層クラスのクラス階層を示している。TProtocolLayerクラス151は、1つのプロトコル・インプリメンテ

ーションのすべての層の基本クラスとして機能する。

【0043】TProtocolLayerクラス151は、各層で実施されるTransmit、Connect、Receive、Disconnectなどの関数用のメソッドを含んでいる。これらのメソッドについては、以下の表3に詳しく示す。

【0044】TCP/IPなどのプロトコル用の具象クラスは、これらの層オブジェクトから派生し、その具体

Dispatch
Transmit
DispatchEvent
TransmitEvent
ReceiveEvent
CancelReceiveEvent
InstantiateInterface
GetUpperProtocol
GetLowerProtocol
Bind
Unbind
Connect
ReceiveConnection
Disconnect
GetPacketQueue
ReceiveDisconnect
AcceptConnection
RejectConnection
Listen
ListenForConnection
SendRelease
ReceiveRelease
演算子<<=
演算子>>=

的なプロトコル層の意味を取り入れている。

【0045】それぞれの子オブジェクト153～161は、これらのメソッドを継承し、特定のプロトコル層に該当する場合にはそのメソッドを指定変更する。

【0046】表3

TProtocolLayerクラスの主な関数を以下に示す。

【0047】

—インバウンド・パケットを上位層にディスパッチする
—アウトバウンド・パケットを下位層に伝送する
—事象を上位層にディスパッチする
—事象を下位層に伝送する
—事象の報告を可能にする
—事象の報告を取り消す
—層オブジェクトからインタフェース・オブジェクトを作成する
—次の上位層へのポインタを獲得する
—次の下位層へのポインタを獲得する
—プロトコル・スタックをバインドする
—プロトコル・スタックをアンバインドする
—接続を確立しようという試みを開始する
—接続を確立しようという試みを待つ
—接続を終了する
—インバウンド・データ・パケットが得られるパケット待ち行列へのポインタを返す
—切断を待つ
—接続開始を受け入れる
—接続を確立しようという試みを拒否する
—接続を確立しようという試みを聴取する
—接続を確立しようという試みを聴取する
—正常解放開始=>送信すべきデータはこれ以上ない
—正常解放開始の肯定応答受信
—TProtocolLayerオブジェクトをデータ・ストリームにマーシャルするための演算子
—TProtocolLayerオブジェクトをデータ・ストリームにアンマーシャルするための演算子

【0048】5. プロトコル状態マシン

ここでは、接続指向操作と無接続操作の両方について可能なプロトコル・インタフェース・オブジェクトの状態について説明する。プロトコル状態マシンを図6および図7に示す。これらの状態マシンは、プロトコルで典型的な状態遷移を示しているが、プロトコルのインプリメンテーションによっては異なる選択も可能である。しかし、この状態マシンは、アプリケーションの可搬性を確実にし、プロトコルの独立性を可能にする。プロトコルの独立性を必要とするアプリケーションでは以下に説明する状態マシンを使用し、プロトコルの独立性をサポート

40 トするプロトコルによって状態マシンが実現されるはずである。

【0049】特定のプロトコル・インタフェース層オブジェクト用のプロトコル端点は、以下に記載する状態のいずれかになるはずである。これらの状態は通常、アプリケーション状態である。層オブジェクトは、特定のプロトコルによって定義される状態マシンに従う。通常、これらの状態は、ユーザが各種の「端点」状態で行うことができる有効な呼出しと、アプリケーションの作成方法を示すためのものである。層状態は、層の意味と状態によって制御される。

【0050】未初期設定状態201は、端点の最終状態でもある端点の開始状態を定義するものである。これは、アクセス定義が作成される前の状態である。

【0051】TAccessDefinitionオブジェクトが作成されると(202)、端点は初期設定済みと呼ばれる。初期設定済み状態203では、Set操作を使用してインタフェース・オブジェクトを構築し初期設定できるはずである。Set操作はインタフェース・オブジェクトでローカルにキャッシュされるので、設定値の妥当性検査を行う余地はほとんどない。すなわち、InstantiateDefinitionメソッドによりプロトコル層オブジェクトが作成される前に、インタフェース・オブジェクト上で送信/受信容量を設定することができる。このような容量への制限に関する情報は層オブジェクトだけが把握しているので、このような値をインタフェース上で妥当性検査することは不可能である。というのは、AccessDefinitionオブジェクト上でInstantiateDefinitionメソッドが呼び出されるまで、層オブジェクトが存在しないからである。TAccessDefinitionオブジェクト204を破棄するよう要求すると、端点が初期設定済み状態203から未初期設定状態201に移行する。

【0052】TAccessDefinitionオブジェクト上でインスタンス化要求206を行うと、端点が未結合状態207に移行する。未結合状態207は、層オブジェクトがインスタンス化された直後に発生する状態を定義する。値を修正するためにGet/Set操作を出すことができる。このような要求はもはやインタフェース・オブジェクトにキャッシュされないが、対応する層オブジェクトに送られて、処理され格納される。TAccessDefinitionオブジェクト上でインスタンス解除要求208を行うと、端点が初期設定済み状態203に移行する。

【0053】ローカル・アドレスをバインドするためにBind操作210が出されると、端点が未結合状態207から結合状態209に移行する。無接続モードのデータ転送の端点は、「結合」状態になるとデータの送受信を開始することができる。結合状態209でスタックに対してアンバインド要求212が出されると、端点が未結合状態207に戻る。

【0054】Listen要求214が出されると、端点が結合状態209から聴取状態213に移行する。プロトコル・スタックは、ユーザ指定の待ち行列サイズを使い尽くすまでそのローカル名に関する着信接続要求を受け入れる。着信接続216により、新しい活動プロトコル・スタックが作成される。

【0055】活動端点でConnect要求220が出されると、端点が結合状態209から接続(クライアント)状態219に移行する。図7に示すように、無接続モードのサービスを使用する端点は、接続要求が正常に行われた後で「データ転送」状態225に入る。受動端点の場合、着信接続要求を受信すると、プロトコル・スタック

は、層とインタフェース・オブジェクトのコピーと、受信した接続要求用の新しいTAccessDefinitionを作成する。新たに作成された端点は、その後、接続(サーバ)状態221になる。矢印は、聴取状態から接続(サーバ)状態まで点線上にある。アプリケーションは、接続を受け入れるか、または接続を拒否するかを選択することができる。AcceptConnectionでは、端点がデータ転送状態225になるはずである。接続要求226が拒否された場合、端点は非活動状態229に移行する。

【0056】新たに作成したスタックが接続要求222を完了後、端点はデータ転送状態225に入る。ローカル・アプリケーションからDisconnect要求228を受信するか、または接続パートナーによって接続が終了された後に、接続端点が結合状態209に移行する。ただし、このような場合、アプリケーションはReceiveDisconnect要求を出して終了データを受信することに留意されたい。

【0057】着信接続要求がアプリケーションによって拒否されると、端点は非活動状態229に移行する。次に、TAccessDefinition破棄操作230を出すことによって、端点が破棄される。

【0058】6. 例

ここでは、TCP/IPなどのネットワーク・プロトコルに本発明のオブジェクト指向モデルを使用する方法の例を示す。TCP/IPインプリメンテーションは、トランスポート層と、ネットワーク層と、TCP/IPファミリー層とを含む。さらに、ネットワーキング・サブシステムは、システム内に常駐するTDatalink層オブジェクトを含み、TCP/IPのFamilyLayerオブジェクトはTDatalink層にアクセスする手段を備えている。ただし、この実施例では、TDatalink層オブジェクトはTCP/IP、SNA、NetBIOSなど、すべてのネットワーク・プロトコル・スタックに共通し、TDatalink層オブジェクトはTProtocolLayerクラス・オブジェクトから派生することに留意されたい。

【0059】TCP/IPインタフェース(API)クラスは、TProtocolInterfaceクラスから派生している。図8は、TCP/IPインプリメンテーションの一部のオブジェクトのクラス階層を示している。前述のように、TProtocolInterfaceとTProtocolLayerは、APIおよびプロトコル・インプリメンテーション機能を提供するMProtocolServiceクラスの子クラスである。TTCPTINF251と、TTCPTNINF253と、TTCPTFINF255の各オブジェクト(まとめてTTCPTXINFと呼ぶ)は、TCP/IPプロトコル用のTTransportInterfaceクラス、TNetworkInterfaceクラス、TFamilyInterfaceクラスを表すTProtocolInterfaceの具象クラスの例である。ただし、TCP/IPプロトコルには「セッション」という概念がないので、TSessionLayerクラスのインスタンスがないことに留意されたい。同様に、TTCPTIMP257と、TTCPTNIMP

259と、TTCPFIMP261の各オブジェクト（まとめてTTCPIXIMPと呼ぶ）は、TCP/IPプロトコル用のTTra
nsportInterfaceクラス、TNetworkInterfaceクラス、TF
amilyInterfaceクラスのインスタンスである。

【0060】前述のように、TDataLinkLayer161は、
この実施例のすべてのネットワーク・プロトコル用のイン
プリメンテーション・オブジェクトであり、TProtoco
lLayerクラス151のサブクラスである。

【0061】また、IPアドレスを渡すことができるよ
うに、アプリケーションにはTTCPPProtocolAddressクラス
が用意されている。TTCPPProtocolAddressは、4バイ
トのIPアドレスと、2バイトのポート・アドレスとを
含むことになる。

【0062】TCP/IPプロトコルにアクセスするた
めにアプリケーションが必要とするプロセスについて
は、図9を参照して以下に説明する。ほとんど場合、
「アプリケーション」は、ユーザ・レベル・アプリケー
ションがAPI呼出しを行うBSDソケット・インタフ
ェースなどの通信API層になる可能性が最も高い。ソ
ケット・インタフェースにより、アプリケーション開発
者は、ネットワーク・プロトコル・オブジェクトの特定
の構文を把握する必要がなくなるはずである。しかし、
「アプリケーション」は、本発明のオブジェクト・モデ
ルの名前に精通したオペレーティング・システム・サー
ビスにもなりうる。以下の例では、アプリケーションが
トランスポート層にアクセスする。アプリケーションは
どの層にもアクセスすることができる。ネットワーク層
に直接送話することも可能である。現在、ユーザ・ア
プリケーションがまったくないことが一般的である。これ
らのステップは、TCP/IPプロトコルの手続き上の
インプリメンテーションのものと高レベルで非常によく
似ている。しかし、それは、本発明のプロトコル・イン
タフェース・モデルを使用して、オブジェクト指向のや
り方で実現されている。

【0063】ステップ301では、TCP/IPプロト
コルにアクセスするために通信端点が作成される。これ
は、まずTAccessDefinitionオブジェクトを作成するこ
とによって行われる。たとえば、C++を使用して「新
しいTAccessDefinition」が構築される。次に、TAccess
Definition内のメソッドを使用して、TTCPIXIMPインタフ
ェース・オブジェクトが作成され、AccessDefinitionに
追加される。その後、TAccessDefinitionオブジェクト
上でInstantiateProtocolメソッドが呼び出され、次に
それによりTTCPIXIMP層オブジェクトが作成される。

【0064】ステップ303では、ステップ301で作
成された通信端点にアドレスが結合される。これは、用
意されたTTCPPProtocolAddressクラス・オブジェクトか
ら必要なIPアドレスを備えたTTCPPProtocolAddressオ
ブジェクトを作成することによって行われる。次に、そ
のアドレスをバインドするためにTTCPTINF->Bind()メソ

ッドが呼び出される。このステップは、バインドの意味
を含むプロトコル・インプリメンテーション層上でTTC
PIMP->Bind()メソッドを起動することになる。

【0065】次に、ステップ305では、（TTCPTINFオ
ブジェクトで）TTCPTINF->Connect()メソッドを呼び出
して接続要求を開始することによって、アプリケーション
が聴取対等機能に接続する。これにより、（TTCPTIMP
オブジェクトで）TTCPTIMP->Connect()メソッドが起動
され、次にこのメソッドは、下位層、すなわち、TTCPN
I 10 MP->Connect()メソッドとTTCPFIMP->Connectメソッド用
のTTCPNIMPオブジェクトとTTCPFIMPオブジェクトをそれ
ぞれ呼び出すことにより、TCP/IP接続をセットア
ップするための必要なステップを実行する。

【0066】正常接続後、ステップ307では、結果の
プロトコル・スタックによりデータを送受信することが
できる。アプリケーションは、TTCPTINF->Send()メソッ
ドとTTCPTINF->Receive()メソッドを呼び出して、ネッ
トワーク・データを送受信する。TTCPTINF->Send()は次
にTTCPTIMP->Xmit()メソッドを呼び出して、TCPプロ
トコルのデータ伝送を開始する。データは、各プロトコ
ル層のXmit()関数を使用してプロトコル層オブジェクト
からプロトコル層オブジェクトに渡され、通信アダプタ
によりTDataLinkLayerオブジェクトに送達される。同様
に、受信機能の場合は、TDataLinkLayerが物理層からデ
ータを受信し、それを適切なプロトコル・ファミリー層
に渡し、次にその層が適切なスタックに渡す。しかし、
一実施態様では、クライアントからのデータ受信要求が
行われるまでデータが待ち行列化され、データはデータ
待ち行列からクライアントにコピーされる。

【0067】ステップ309では、所望の通信の完了後
に接続が閉じられる。アプリケーションは、TTCPTINF->
Disconnect()メソッドを呼び出して、接続終了を開始す
る。これにより次にTTCPTIMP->Disconnect()が呼び出さ
れ、このメソッドは特定のインプリメンテーションの場
合にファミリー層までそのメソッドを送信する可能性の
あるTTCPTIMPのTCP/IP切断状態マシンを処理す
る。

【0068】最後に、ステップ311では、TAccessDef
initionオブジェクトを削除することによって、端点が
閉じられる。

【0069】図10を参照すると、様々なオブジェクト
とネットワーク層との関係を理解するのに役立つだろ
う。同図の左側には、図8および図9に関連して前述し
たTCP/IP実施例が示されている。TCP/IPア
プリケーション313、315は、いずれもアプリケー
ション層にあるソケットAPI317を介して、以下の
層のオブジェクト指向プロトコル・スタックに連絡す
る。通信端点を作成するためにソケットAPIが使用す
るTProtocolAddressオブジェクトは図示していない。そ
れぞれのTCP/IPアプリケーション313、315
50

ごとに個別の通信端点、すなわち、個別のTProtocolAddressオブジェクトが必要である。

【0070】前述のように、TCP/IPにはセッション層という概念がないので、ソケットAPI 317はトランスポート層内のTTCPTINFオブジェクト251によってやりとりする。前述のように、TAccessDefinitionオブジェクト318は、トランスポート層、ネットワーク層、ファミリー層のTTCPTINFオブジェクト251、253、255を含む。ユーザ・レベル・プロセスがファミリー層のネットワークによりプロトコル・スタックに連絡することは比較的まれなことであるが、主にオペレーティング・システム・サービスにより、これらの層に連絡するためにTTCPTINFオブジェクト253とTTCPTINFオブジェクト255が提供される。

【0071】TTCPTINFオブジェクト251は、TTCPTINF-->Connect()メソッドによりTTCPTIMPオブジェクト257に接続する。これにより、TTCPTIMPオブジェクト259に接続するためのTTCPTIMP-->Connect()メソッドと、TTCPTIMPオブジェクト261に接続するためのTTCPTIMP-->Connect()メソッドが起動される。また、TDataLinkLayerオブジェクト161に接続するTTCPTIMP-->Connect()メソッドが起動される。同図に示す通り、TTCPTIMP257、TTCPTIMP259、TTCPTIMP261、TDataLinkLayer161の各オブジェクトは、それぞれトランスポート層、ネットワーク層、ファミリー層、データリンク層にある。前述のように、送受信メソッドにより、物理層内の物理アダプタ319を介してプロトコル・スタック上でネットワーク・データを送信することができる。

【0072】好ましい実施例では、通信端点が作成され削除されたときに、TDataLinkLayerオブジェクト161とTTCPTIMPオブジェクト261は不変であり、単独である。それぞれの活動端点ごとに、残りのオブジェクト(251、253、255、257、259、318)のそれぞれの個別のスレッドとインスタンスが必要になる。当業者であれば容易に分かるように、何千ものクライアント接続が行われる可能性のあるネットワーク・プロトコル・サーバの場合、このような配置に関連するオーバーヘッドによってパフォーマンス上の問題が発生する可能性がある。以下の動的実行ユニット管理の項で説明するように、本発明は、パフォーマンスを最大にするために実行スレッドを動的に管理するための手段を提供する。

【0073】同図の右側には、Net BIOSのインプリメンテーションが示されている。したがって、本発明は、システムが複数のネットワーク・プロトコルを実行する複数のアプリケーションをサポートするマルチプロトコル環境を企図するものである。図10に示すように、Net BIOSアプリケーション331、333は、ソケットAPI 317または特殊Net BIOS API層335のいずれかを介して下位層とやりとりす

ることができる。したがって、本発明は、本発明のオブジェクト指向プロトコル・スタックにアクセス可能な複数のアプリケーションも企図するものである。このようなアプリケーションは、オブジェクト・ベースまたは手続きベースのユーザ・レベル・アプリケーションまたはオペレーティング・システム・レベル・プロセスにすることができる。

【0074】前述のように、同様のプロセスを使用してNet BIOSスタックが確立される。ただし、Net BIOSにはセッション層という意味がないので、API層317、335はセッション層のTNSINFオブジェクト337に接続することに留意されたい。Net BIOSインタフェース・オブジェクトであるTNSINF337、TNTINF339、TNNINF341、TNFINF343と、Net BIOSプロトコル層オブジェクトであるTNSIMP347、TNTIMP349、TNNIMP351、TNFIMP353が作成された後、TNSINFオブジェクト337によりプロトコル層オブジェクトを介してTDataLinkLayerオブジェクト161まで通信経路が確立される。TCP/IP実施例のように、TDataLinkLayerオブジェクト161とTNFIMPオブジェクト353は不変かつ単独であり、それぞれの活動通信端点ごとに残りのオブジェクトのそれぞれの個別のインスタンスが作成され削除される。

【0075】本発明では、TDataLinkLayer161に結合された複数のアダプタ319、321に対処することができる。上位層は通信端点までの適切な経路指定を行うので、異なるネットワーク・プロトコルに対して同時にこれらのアダプタを使用することができる。さらに、これらのアダプタは、イーサネットとトークン・リングなど、異なるタイプのものにすることができるので、サーバは各種のネットワークからの要求に対応することができる。

【0076】7. プロトコル・ユーティリティ・クラス
このクラスは、TProtocolInterfaceクラスとTProtocolLayerクラスが使用するものである。このクラスのほとんどは、インタフェース・クラスとインプリメンテーション・クラスの様々なメソッドのパラメータとして機能する。このクラスは、提案したモデルの一般的な特徴を強調するものである。

【0077】ここでは、プロトコル・インタフェースが使用する重要なクラスの一部について説明する。

【0078】a. TProtocolInfoクラス

このクラスは、サービス・タイプ、最大メッセージ・サイズ、優先データ長など、そのプロトコルの様々な特性を識別するものである。このクラスが提供する重要なメソッドの一部を以下に示す。

【0079】これらは、端点の作成時に端点の特性を指定するために使用する。

【0080】

25

26

GetPSDUSize	—プロトコル・サービス・データ・ユニットの最大サイズを獲得する
SetPSDUSize	—プロトコル・サービス・データ・ユニットの最大サイズを設定する
GetEPSDUSize	—優先プロトコル・サービス・データ・ユニットの最大サイズを獲得する
PSDUSize	—優先プロトコル・サービス・データ・ユニットの最大サイズを設定する
GetConnectionDataSize	—接続データの最大サイズを獲得する
SetConnectDataSize	—接続データの最大サイズを設定する
GetDisconnectDataSize	—切断データの最大サイズを獲得する
SetDisconnectDataSize	—切断データの最大サイズを設定する
GetDisconnectDataSize	—切断データの最大サイズを獲得する
SetDisconnectDataSize	—切断データの最大サイズを設定する
GetServiceType	—接続／無接続などのサービス・タイプを獲得する
SetServiceType	—サービス・タイプを設定する
SetServiceType	—サービス・タイプを設定する
SetServiceFlags	—サービス・フラグを設定する

【0081】b. TProtocolOptionsクラス

このクラスは、サービス品質を含む、プロトコル固有のオプションを定義するために使用する。しかし、この基本クラスは、特定のサービス品質を含んでいない。具象

20 クラスがTProtocolOptionsから派生し、その固有のオプションを含むものと思われる。

【0082】

GetLingerTime	—プロトコル・リング時間を獲得する
SetLingerTime	—プロトコル・リング時間を設定する
GetSendbufSize	—プロトコル送信バッファ・サイズを獲得する
SetSendbufSize	—プロトコル送信バッファ・サイズを設定する
GetRcvbufSize	—プロトコル受信バッファ・サイズを獲得する
SetRcvbufSize	—プロトコル受信バッファ・サイズを設定する

【0083】c. TSendModifiersクラス

TSendModifiersクラスは、TProtocolInterface::SendMessage()とTProtocolLayer::Xmit()メソッドにおけるネットワーク送信機能に関連するフラグを修飾するものであ

30 る。送信タイムアウトのサポートの他に、送信機能を左右しうる指示は以下の通りである。

【0084】

kPush	—即時処理を要求する
kEdnOfMessage	—メッセージの終わりをマークする
kExpeditedData	—データを優先データとして扱う
kNotify	—クライアント・バッファが使用可能なときに送信側に通知する
kSendAll	—「n」バイトがすべてバッファされるまでブロックする
kNoFragmentation	—データを断片化しない

【0085】このクラスでサポートされる重要な関数の一部を以下に示す。

【0086】

GetSendModifier	—SendModifier用の値を獲得する
SetSendModifier	—SendModifierをONに設定する
ClearSendModifier	—SendModifierをOFFに設定する
GetTimeout	—送信タイムアウトを獲得する
SetTimeout	—送信タイムアウトを設定する

【0087】d. TReceiveModifiersクラス

このクラスは、TProtocolInterface::Receive()関数内

の受信フラグを定義するために使用する。

50 【0088】このクラスは、ネットワーク・データを受

信しながら、フラグとタイムアウトを設定するためのメソッドと定義を含む。以下の通りである。

【0089】

kPeek	－ピーク・ユーザ・データ
kExpeditedData	－優先データを受信する
kReceiveAll	－「n」バイトが受信されるまでブロックする
kBroadcastData	－同報データグラムを受信する
kDiscardBufferOverflow	－受信バッファからオーバーフローするメッセージの一部を破棄する
kDatagramAny	－通常データグラムまたは同報データグラムのいずれかを受信する

【0090】このクラスの重要な関数の一部を以下に示す。

【0091】

GetReceiveModifier	－ReceiveModifier用の値を獲得する
SetReceiveModifier	－ReceiveModifierをONに設定する
ClearReceiveModifier	－ReceiveModifierをOFFに設定する
GetTimeout	－受信タイムアウトを獲得する
SetTimeout	－受信タイムアウトを設定する

【0092】e. SendCompletionクラス

示し、送信バイトの総数も示す。状況指示は以下の通り

ネットワーク・データ送信操作に応答してSendCompletion

20 である。

オブジェクトが返される。これは、送信機能の状況を

【0093】

kNormal	－正常完了
kTimeout	－送信タイムアウト
kCancelled	－アプリケーションが送信要求を取り消した

【0094】このクラスの重要なメソッドの一部を以下に示す。

【0095】

GetStatus	－SendCompletion状況を獲得する
SetStatus	－SendCompletion状況を設定する
GetBytesTransferred	－送信するクライアント・データのバイト数を獲得する
SetBytesTransferred	－送信するクライアント・データのバイト数を設定する

【0096】f. TReceiveCompletionクラス

況と、データ受信バイトの総数を示す。状況指示は以下の通りである。

ネットワーク・データ受信要求に応答してReceiveCompletionオブジェクトが返される。これは、受信機能の状

【0097】

kNormal	－正常完了
kPushed	－送信側がデータを「プッシュした」
kNoMoreData	－ストリームが終了するかまたは受信パイプが閉じられた
kEndOfMessage	－メッセージの終わりが検出された
kExpeditedData	－優先データを受信した
kTimeout	－受信タイムアウト
kMessageTruncated	－部分メッセージであり、残りは破棄される
kCancelled	－取消し要求が処理された
kMore	－追加データの受信準備ができている

【0098】このクラスの重要な関数の一部を以下に示す。

【0099】

GetStatus	－ReceiveCompletion状況を獲得する
SetStatus	－ReceiveCompletion状況を設定する
GetBytesTransferred	－受信するクライアント・データのバイト数

SetBytesTransferred

【0100】ネットワーク事象管理

ここでは、複数のネットワーク・プロトコル・スタック上で複数のネットワーク事象にアクセスし、それを様々なプロトコル層で格納するという問題に対するオブジェクト指向の解決策を開示する。本発明では、OSIネットワーク・プロトコル層モデルに基づくすべてのネットワーク事象の単一かつ一貫した1組のクラス定義を提供する。各総称OSI層ごとに事象が定義されているので、これらの層の各インプリメンテーションは追加のサブクラスを定義することによって、追加の層固有の事象を加えることを選択できるはずである。この解決策は、オブジェクト指向モデルなので、OO技術で多様性と継承を完全に利用する。

【0101】OSI層に基づく事象の分類により、プロトコル層で格納され報告される事象が明確にカテゴリー化される。このカテゴリー化により、アプリケーションにとって重要であれば、特定の層について事象を取り出すことができる。このような事象は、データが受信済みで受信に使用可能である、接続が打ち切られたなど、確立されているクライアントの通信セッションの条件を記述することができる。クライアントは、単一の呼出しを使用して、複数のプロトコル・スタックのいずれのプロトコル層上でもこれらの非同期事象を監視することができる。本発明は、アプリケーションが個々の端点事象を探する必要がなくなるように、TCP/IPやNetBIOSなどの複数のプロトコルによる事象報告に及ぶ。さらに、ネットワーク事象機構により、層オブジェクト間でネットワーク事象を送受信するためにネットワーク・プロトコル層間でのやりとりが容易になる。

【0102】1. 事象の説明

事象は、TProtocolEventという基本クラスを使用して定義される。図11は、様々な事象オブジェクト間のクラス階層を示している。図11は図2に似ているが、TEventDefinitionクラス103のサブクラスであるTProtocolEventクラス401も示している。TProtocolEventクラス401は、列挙されたクラスの事象と、すべてのOSI層で一般に使用される1組の事象と、個々のプロトコル層事象用の値の予約範囲を含む。TProtocolEventクラスは、事象とメンバーシップの質問を設定し獲得するためのメソッドを有する。一実施例のネットワーク事象はビット・ベクトルとして設定されている。このクラスでは、GetCntメソッドがこのオブジェクト内に現在ある事象の数を返す。これは、この場合のオプション事象のリストを指定する典型的な方法であり、一般に使用するもう1つの方法は、ビット・マスクを使用する方法である。オブジェクト内にすべてのデータを隠すというオブジェクト指向技術の能力のために、単一オブジェクトだ

を獲得する

一受信するクライアント・データのバイト数を設定する

けが必要であり、そのオブジェクトがそれらを獲得するためのメソッドを提供する。

【0103】特定のネットワーク・プロトコルによって、TProtocolEventクラスで事象値を再定義することによって、プロトコル事象を定義することができる。TProtocolEventの重要なメソッドの一部は、識別されたインデックスの事象を返すGetEventと、リスト内の特定の事象（メンバーシップ）を探索するHasAと、オブジェクト内の事象の数を返すGetCntと、端点を示すアクセス定義を返すGetAccessDefinitionと、端点用のアクセス定義を設定するSetAccessDefinitionと、TProtocolEventオブジェクト内の単一事象を設定するSetEventである。

【0104】2. OSI層インプリメンテーションへの事象インタフェース

上記のプロトコル・インタフェース・モデルの項で述べたように、好ましい実施例ではネットワーク・プロトコル層が一連のTProtocolLayerオブジェクトとして実現され、それぞれが特定のOSI層を表している。これらのオブジェクトは互いに連鎖され、プロトコル・スタックを完成させている。たとえば、TCP/IPスタックは、トランスポート層、ネットワーク層、データリンク層用のオブジェクトを備えることができる。各プロトコル層オブジェクトは2つの事象インタフェース、すなわち、TProtocolLayerオブジェクトに定義されたReceiveEventインタフェースとCancelEventインタフェースとを有する。これらの層では、上位層のオブジェクトに事象を渡すためにDispatchEventメソッドを使用し、下位プロトコル層に事象を送信するためにTransmitEventメソッドを使用する。適切な層に事象を格納することは、プロトコル・スタックのインプリメンテーションの責任である。

【0105】TProtocolLayerクラスの事象関連メソッドは、上位層に事象をディスパッチするDispatchEventと、下位層に事象要求待ち行列を伝送するTransmitEventと、EventRequestQueueにTProtocolEventオブジェクトを追加することにより事象（複数可）を報告するReceiveEventと、EventRequestQueueを無視するCancelReceiveEventである。

【0106】3. アプリケーションによる事象監視
それぞれのネットワーク・プロトコル層にオブジェクト・ベースのAPI用の基本クラスを提供するTProtocolInterfaceクラス・オブジェクトは、ネットワーク事象を受信するために端点が使用できる機能を含んでいる。TProtocolInterfaceクラスの事象関連機能は、特定の通信端点用にTProtocolEventオブジェクトに設定された事象（複数可）を受信するReceiveEventメソッドである。

【0107】アプリケーションが複数の通信端点での事

象の監視を必要とするときは、TEventDefinitionオブジェクトを使用する。TEventDefinitionオブジェクトは、事象を監視するためのオブジェクト端点として機能する。このオブジェクトのインスタンスは、アプリケーションの要求に応じて作成され削除される。TEventDefinitionオブジェクトの主なメソッドは、TEventDefinitionオブジェクトのインスタンスをインスタンス化するInstantiateDefinitionと、TEventDefinitionオブジェクトのインスタンスを破棄するDeInstantiateDefinitionと、TProtocolEventオブジェクトのアレイまたは待ち行列で複数のスタックからの事象を受信するReceiveEventAnyと、保留事象要求を取り消すCancelRequestである。

【0108】a. 1つの通信端点での事象の受信
アプリケーションは、特定のネットワーク層用のTProtocolInterfaceクラスのインスタンスでReceiveEventメソッドを使用して、特定の端点で事象を受信することができる。ただし、端点はそのプロトコル用のTProtocolInterfaceオブジェクトを使用してプロトコルにアクセスすることに留意されたい。アプリケーション・プログラムは、必要な事象をTProtocolEventオブジェクトに設定し、次にReceiveEventメソッドを呼び出す。TProtocolInterfaceオブジェクトは、事象が発生していればその事象を含むTProtocolEventオブジェクトを返す。

【0109】1つの通信端点で事象を受信するプロセスを図12の流れ図に示す。この例では、トランスポート層が一番上の層であり、事象はネットワーク層に格納され、アプリケーションからの要求はトランスポート層に対して行われる。

【0110】もう1つの代替方法は、事象を受信するためにその層への直接の事象インタフェースをセットアップする方法である。しかし、このプロセスでは、あまり有利ではない事象についてもすべての層をそのように処置しなければならない。1つの点から獲得する方が良好なので、クライアントは選択基準として事象ベクトルを使用してすべての層から事象を獲得するように呼出しを行う。

【0111】図12は、単一端点での事象受信機構を示している。端点で1つまたは多数の事象を受信するための要求は、TProtocolInterface::ReceiveEvent()メソッド411を使用してインタフェース・オブジェクトに対して行われる。結果的に端点から報告されるTProtocolEventsを含むProtocolEventQueue413が内部で作成される。次に、TProtocolLayer::ReceiveEvent()メソッド415を使用して対応する層オブジェクトにその要求が送られる。同図は、層オブジェクト間でどのように要求が送られるかを示している。TProtocolEventQueueはReceiveEvent要求内のパラメータとして送られる。プロトコル層オブジェクトは、事象が発生するたびに、TProtocolEventQueueを無効にするTProtocolLayer::CancelEvent()要求を受信する前に、この待ち行列にTProtocolEve

ntを待ち行列化する。要求された事象に応じて、TxxTransportLayerは、TProtocolLayerクラスのTransmitEvent()メソッドを使用してTxxNetworkLayerに要求を送ることができる(417)。同様に、事象要求が下位層に到達する必要がある場合には、TxxNetworkLayerは、要求をTxxFamilyLayerに送ることもできる(419)。

【0112】事象が非同期に発生する場合、TProtocolLayerオブジェクトは、DispatchEvent()を使用してその事象を上位層に報告することができる(421)。次にネットワーク層は、この事象をTxxTransportLayerに報告する(423)。TxxTransportLayerは報告された事象をProtocolEventQueueに待ち行列化する(415)。次に、事象は呼出し側に報告される(411)。

【0113】b. 複数の通信端点での事象の受信
アプリケーションは、TEventDefinitionオブジェクトを使用することにより、複数の通信端点の事象のための端点を作成することができる。図11に示すように、TEventDefinitionクラス103はTNetworkDefinitionクラス100から派生している。TEventDefinitionオブジェクト103は、任意のプロトコルで任意の数の端点の事象を監視するための端点として機能する。アプリケーション・プログラムにとって関心のある各通信端点ごとに、必要な数のProtocolEventsがTProtocolEventオブジェクトに設定される。すなわち、その事象が監視対象となるすべての端点について、2タプル(端点、プロトコル事象)の対が形成される。その後、アプリケーションは、様々な端点から要求された事象の組を渡して、TEventDefinitionオブジェクトのReceiveEventAny()メソッドへの呼出しを行う。ReceiveEventAnyは、TProtocolEventsの待ち行列に事象を入れて復帰するか、またはタイムアウトする。ただし、すべての通信端点はTAccessDefinitionクラスを使用して作成されることに留意されたい。

【0114】図13は、複数の端点での事象受信の流れを示している。

【0115】図13は、複数の端点での事象受信機構を示している。アプリケーションは、事象端点として機能するTEventDefinitionオブジェクトを作成する(451)。次にアプリケーションは、端点当たり1つずつ、TProtocolEventオブジェクトのアレイを作成する(453)。端点での要求された事象のアレイがパラメータとしてTEventDefinition::ReceiveEventAny()に渡される(455)。ReceiveEventAny()メソッドは、TProtocolEventQueueを作成し(457)、事象が探索されるすべての端点に待ち行列を送る(461)。ReceiveEvent要求を介して要求を受信する(463)TxxProtocolLayerは、それより下の層に事象要求を伝送することができる。これは、どの端点についても同じである(465)。TxxProtocolLayerは、事象を非同期に受信すると、上位層にその事象をディスパッチするか、またはTxxProtocolEventQueueで事象を報告することができる。事

象がTProtocolEventQueueに入ると、すべての関連端点のTxProtocolLayerにTxProtocolLayer::CancelEvent()が送られる。次に、ReceiveEventAnyメソッド455は、TProtocolEventQueueからすべてのTProtocolEventsを取り出し(459)、それを呼出し側に報告する。報告する事象がまったくない場合には、待ち行列が空であるはずである。

【0116】いずれかの端点から応答が受信された後、すべての端点はCancelEvent要求を受け取り、ProtocolEventQueueオブジェクトの寿命が終了したことを示す。その要求が取り消された場合でも、事象は依然としてプロトコル層に保管される。クライアントは、次に新しい事象を獲得するときに、別のReceiveEventとともに戻ることができる。この実施例では、クライアントからのブル・モデルを使用するが、クライアントにとって便利なきにクライアントが事象を獲得することを意味する。呼出し側には収集された1組のProtocolEventsが返される。

【0117】動的実行ユニット管理

マルチタスキング・システムでは、TCP/IPなどのネットワーク・プロトコル・スタックは、クライアント／サーバ・モデルを使用して効率よく実現することができる。サーバは、多数のクライアントをサポートするためにマルチタスキング技法を使用してユーザ・レベル・タスクにすることができる。多数のクライアントをサポートする際に十分なパフォーマンスを保証するため、固定した1組のサーバ資源を特定のプロセスに事前割振りすることが必要な場合も多い。重要なサーバ資源の1つは、割り振られたサーバ実行ユニットの数である。OS/2やWindows NTなどの最新のオペレーティング・システムでは、実行の基本ユニットはスレッドである。現在の多くのシステムは「軽量」スレッド(高速スレッド切替えを意味する)を備えているが、残念ながら、スレッド作成／削除は、何千ものクライアントをサポートするために拡大したときに本発明が提案するようなネットワーク・プロトコル・サーバのパフォーマンス要件をサポートできるほど十分「軽量」ではない。

【0118】さらに、スレッドの数が増すにつれて、システム、すなわち、サーバのパフォーマンスは大幅に低下する。スレッド切替えのオーバーヘッドは、システム内に割り振られたフットプリントに加え、スレッドの数が増すにつれて比例して増加する傾向がある。これは、ネットワーク・セッション下の全二重通信で2つのスレッドに入れてデータの送受信を実行するなど、同じクライアントからの複数の同時要求をサポートする必要があるようなネットワーク・サーバの場合にさらに複雑になる。その結果、複数の同時クライアント対応専用のサーバ内のスレッドの数は非常に大きくなる可能性がある。したがって、スレッドなどのサーバ実行ユニットを効率よく管理することは、十分なパフォーマンスを備えた多

数のクライアントをサポートするサーバの能力にとって重要なことである。

【0119】ここでは、ネットワーク・プロトコル・サーバが実行スレッドの数を動的に管理するための方法を開示する。この方法は、多数のクライアントに対応するのに必要なスレッド資源を低減する。個々のクライアントがサーバ割振りのスレッド資源をすべて消費してしまわないように、進入制御が設けられている。同時に、サーバ・スレッド資源を待って特定のクライアントをいつまでも停止させておくことはない。

【0120】1. サーバ実行ユニット管理

サーバ・スレッドは、サーバ・スレッド・プールで作成され管理される。まずサーバが始動すると、サーバ管理スレッドが作成される。好ましいオブジェクト指向の実施例では、これは、以下に記載するクライアント・ポート・プールとネットワーク・サーバ・スレッド・プールを管理するためのメソッドを含むTNetworkServerクラス・オブジェクトをインスタンス化することによって行われる。サーバ管理スレッドは、スレッド・プールでのサーバ・スレッドの作成または削除を調整することにより、サーバ・スレッド・プールの管理を担当する。サーバ管理スレッドは通常、休眠状態である。これは、要求信号とタイマによって定期的に起こされる。

【0121】すべてのクライアント要求サービスは、最初にサーバとのセッションを確立しなければならない。セッション要求は、session_portという単一のサーバ通信端点に送られる。サーバとの通信セッションが確立した後、各クライアントにはclient_portという固有の通信端点が割り当てられる。好ましい実施例では、TAccessDefinitionオブジェクトとサーバ側で割り当てられた各client_portとの間に1対1の対応が存在する。

【0122】クライアントとサーバは、RPC機構を使用して互いにやりとりすることができる。遠隔プロシージャ呼出し(RPC)機構は、後述するように必要な要求経路指定と進入制御を行う。RPC機構は、基礎となるオペレーティング・システムの内部プロセス通信の内部プロシージャ呼出し(IPC)サービスの上に構築することができる。サーバ・タスクの位置の特定に加え、基本RPCプリミティブは以下の通りである。

【0123】クライアント側

○ SendRequest (server_port(セッションまたはクライアント), request_data, reply_data)

サーバ側

○ ReceiveRequest (request_data)

○ SendReply (reply_data)

【0124】すべてのクライアントのclient_portはclient_portプールに収集される。その割当てclient_portを使用して、クライアントはサーバにネットワーク要求を出すことができる。サーバは、client_portプールからの要求を管理し、スレッド・プール内のスレッドを任

意の数のclient_portプールから受け取ったサービス要求に割り当てる。図14は、オブジェクト指向ネットワークの第1の実施例でのclient_portプールとスレッド・プールの制御及び管理の流れを示しているが、クライアントおよびサーバ・アプリケーション・オブジェクトは対称マルチプロセッサ（SMP）などのマルチプロセッサ構成の各種プロセッサに割り振られたメモリ内に常駐し、両者はオブジェクト指向RPCインタフェースを介してやりとりする。

【0125】複数のクライアント・マシン500は、ネットワーク505によりサーバ・マシン503に連絡する。それぞれのクライアント・マシン500は、それぞれがクライアント要求を行うことができる複数のクライアント・アプリケーション・オブジェクト507を常駐させておくことができる。図示の通り、クライアント・オブジェクト507は、RPC/API層509の適切なRPC/APIオブジェクトに自分の要求を送り、その層は異なるアドレス空間で動作するIPCオブジェクト511を使用してサーバへの接続を行う。プロトコル・スタックは、ネットワーク505への通信経路を確立する。この実施例によれば、クライアント・マシンまたはクライアント・タスクが要求されたサーバでその許容量の通信スレッドを超えているかどうかを判定するために、以下に詳述するスレッド・カウント・オブジェクトがプロトコル・スタックに追加されている。超えていない場合、通信経路は作成されず、クライアント・プロセスは使用可能なスレッドを待つように指示されるか、またはプロトコル・スタックが単にクライアント・プロセスへのサービスを拒否することになる。同図にはサーバが1つだけ示されているが、スレッド・カウント・オブジェクトは、ネットワーク内の複数のサーバ・マシンでクライアント口座を追跡することができる。

【0126】サーバへの通信経路が確立されると想定して、プロトコル・スタック513は、サーバ側の通信端点であるセッション・ポートを割り当てるセッション・ポート・オブジェクト519に送るクライアント要求を受信する。クライアント要求には、クライアント・プール・オブジェクト521からクライアント・ポートが割り当てられる。クライアント・ポートは、サーバ側のスレッド・プール522に使用可能な十分なスレッドがある場合にクライアント・プロトコル・スタックに返されるクライアント通信端点である。セッション・ポート割り振りと、クライアント・ポート・プールと、スレッド・プールとを管理する1組のネットワーク・プロトコル・サーバ・オブジェクトについては、以下に詳述する。スレッドと、クライアント・ポートと、セッション・ポートが割り振られると、サーバ・セットのRPC/APIオブジェクト515と、上方向のサーバ・サービス517への通信を続行することができる。

【0127】動的実行ユニット管理は、手続きベースの

システムでも実行することができる。また、図15に示すように、この機構は、メモリがクライアント空間531とサーバ空間533に分割される単一プロセッサ・マシンや、複数のクライアント・タスク535、537、539がサーバ・タスク540と同じマシンで動作する環境に応用することもできる。クライアント・タスクは、その要求をクライアント側のRPCライブラリ・インタフェース541を介してサーバ・タスクに送る。要求はまず、サーバ側のRPCライブラリ543によってサーバ側で処理される。次に、セッション・ポート・プロセス545によってセッション・ポートが割り振られ、クライアント・ポート・プール547からクライアント・ポートが割り振られる。プロトコル・スタック551がネットワーク553へのクライアント要求を処理できるように、スレッド・プール549からスレッドが割り当てられる。

【0128】プール内のスレッドの数は、複数のシステム構成変数によって制御される。これらの変数の修正は、ユーザ構成プログラムを使ってシステム管理者が行うことができる。

【0129】○（MinThreads）—これは、クライアント要求に対応するために予約されたスレッドの最小数である。この限界は、典型的なサーバ・ロードのクライアントをサポートするために使用するスレッドの最小数を維持するためのものである。

【0130】○（MaxThreads）—これは、クライアント要求に対応するために予約されたスレッドの最大数である。この限界は、サーバ・システムの過負荷にならずにピーク・ロードのクライアントの対応専用となるスレッド資源に関する上限として使用するものである。

【0131】○（MaxReq）—これは、所与のクライアントに割り振られる同時要求の最大数である。

【0132】○（TotalThreads）—これは、すべてのクライアント要求に対応するために作成されたスレッドの総数である。この値は、必ずMinThreadsとMaxThreadsとの間になる。

【0133】○（ReservedThreads）—これは、すべてのクライアント・セッションをサポートするために予約されたスレッドの総数である。

【0134】○（Unusedthreads）—これは、プール内の未使用スレッドの数である。

【0135】○（Clientthreads）—これは、各クライアントごとに、サーバが対応する活動要求の数である。

【0136】プール内のスレッドの数は、同時に処理される活動クライアント要求の数に基づいて、動的に増加または減少する。

【0137】スレッドの数を管理するために使用するプロセス、すなわち、オブジェクト指向実施例のメソッドについては、図16～19を参照して以下に詳述する。

【0138】図16を参照すると、ステップ575のサ

サーバ初期設定では、<MinThreads>個のスレッドが作成され、スレッド・プールに挿入される。また、このステップでは、(UnusedThreads)と(TotalThreads)が(MinThreads)に設定され、(ReservedThreads)が0に設定される。ステップ577では、新しいクライアントがサーバとのセッションを要求する。サーバ側では、クライアント要求を受信後に、(ReservedThreads) + (MaxReq) <= (MinThreads)であるかどうかを判定するためにステップ579でテストが行われる。この結果が真であれば、既存のクライアントと新しいクライアントを

サポートするために十分なスレッドが存在するので、スレッド・プールを調整するためのアクションは一切行われない。

【0139】ステップ581では、サーバがクライアント・タスクにclient_portを割り当てて返す。ステップ583では、(ReservedThreads)が(MaxReq)だけ増分され、新しいクライアントがセッションを要求するまで待つためにステップ577に戻る。

【0140】ステップ579のテストが否定の場合、クライアント要求が許可された場合に最大数のスレッドを

超えるかどうかを判定するためにステップ585でテストが行われる。(MinThreads) < (ReservedThreads) + (MaxReq) < (MaxThreads)かどうかのテストが行わ

New client request session with server
On the server -

```
If (ReservedThreads) + (MaxReq) <= (MinThreads),
    (ReservedThreads) increment by (MaxReq)
    Assign and return a client_port to the client
If (MinThreads < (ReservedThreads) + (MaxReq) < (MaxThreads)
    Increment (ReservedThreads) by (MaxReq)
    Assign and return a client_port to the client
If (ReservedThreads) + (MaxReq) > (MaxThreads)
    Reject the network request
```

【0144】図17には、クライアント要求の進入制御を管理するためのクライアント側プロセスが示されている。ステップ601では、クライアント・タスクがサーバに割り当てられたクライアント・ポートを使用して新しいネットワーク要求を出す。ステップ603では、クライアント・タスクがサーバ側の許容数のスレッドのうち使用可能な複数のスレッドを備えているかどうかを判定するためにテストが行われる。これを判定するために、(ClientThreads) < (MaxReq)かどうかのテストを使用することができる。備えている場合、ステップ607でクライアント要求はサーバに送られる。ステップ609では、現行クライアント・スレッドの数が増分される。

【0145】サーバ側の許容数のスレッドのすべてが使用中である場合、ステップ605でネットワーク要求が拒否されるか、またはこのクライアントからの同時要求の数が(MaxReq)より少なくなるまで待機状態に入る。

れる。この結果が真であれば、スレッド限界を超えておらず、プロセスはステップ581に移行し、クライアント・タスクにクライアント・ポートを割り当てて返す。ステップ583では、(ReservedThreads)が(MaxReq)だけ増分される。後述するように、この図のプロセスはこの分岐と上記の分岐の場合に同じように見えるが、管理スレッドは、新しいカウント(ReservedThreads)に基づいて次に起こされたときにスレッド・プール内のスレッドを非同期に増加する。

10 【0141】ステップ585のテストが肯定の場合、すなわち、(ReservedThreads) + (MaxReq) > (MaxThreads)である場合、ステップ587でクライアント要求が拒否される。したがって、サーバは、過負荷の状況を回避するためにクライアント・セッション進入制御を管理する。ステップ589では、スレッドが他のクライアント・タスクから解放されるまで、サーバは待ち状態に入り、おそらくクライアントに待ち状態を通知するはずである。あるいは、サーバは、クライアント要求を拒否し、後でクライアントがもう一度試行できるようにする

20 だけである。

【0142】以下の擬似コードは、このプロセスの実現例を示している。

【0143】

待機状態になっているときに、許容スレッドの1つが解放された場合、プロセスはステップ605からステップ607に進むことができる。

40 【0146】図17には、クライアント要求に対応するためにスレッド・プール内のサーバ・スレッドを割り当て、プール内のスレッドの数を調整するためのサーバ側のプロセスも示されている。ステップ611では、クライアント要求の処理に必要な期間中、サーバ・スレッドがクライアント要求に割り当てられる。ステップ613では、(UnusedThreads)変数が減分される。ステップ615では、スレッド・プール内のスレッドの数が許容できるかどうかを判定するためにテストが行われる。

50 ((UnusedThreads) < (MinThreads) & (TotalThreads) < (MaxThreads))かどうかのテストは、このステップを実行するためのテストの1つである。スレッド・プール内の使用可能なスレッドの数が許容できない場合には、スレッド・プール内のスレッドの数を増加するよ

うに管理スレッドに通知される。プロセスはステップ619で終了する。

【0147】以下の擬似コードは、このプロセスの実現

```
Client issues network request to the server
If (ClientThreads) < (MaxReq).
    Send the request to the server.
    Increment (ClientThreads)
else
    Reject the network request
```

【0149】

10

```
Assign server thread to service client request
Decrement (UnusedThreads)
If (UnusedThread) < (MinThread) & (TotalThread) < (MaxThread)
    Increase threads in the thread pool
```

【0150】図18には、サーバがクライアント・ネットワーク要求を完了したときにクライアントとサーバが従うプロセスが示されている。サーバ側では、ステップ625でクライアント要求にどのような処理が必要であってもその処理をサーバが完了している。ステップ627では、プロトコル・スタックを介してクライアント要求への応答が送られる。次に、ステップ629では、サーバ・スレッドがスレッド・プールに返される。次に、ステップ631では、(UnusedThreads)変数が増分される。

【0151】クライアント側では、ステップ635でプロトコル・スタックがサーバからの応答を受け取る。ステップ637では、他のクライアント・タスクが通信スレッドのクライアント割振り分を使用できるようにするために、(ClientThreads)変数が減分される。ステップ639では、(ClientThread)が(MaxReq)限界を上回っているためにサーバに要求を送るために待機している同じクライアントから任意のスレッドを起こすための信号が出される。ステップ641では、プロセスが図17のステップ607に移行する。

【0152】図19には、サーバ管理スレッド・プロセスが示されている。サーバ管理スレッドは、スレッド・プール内の未使用スレッドの数がある程度低い限界より下がったときにスレッド割振りのためにタイマまたは信号によって起こされる。ステップ653では、さらにスレッドを追加する必要があるかどうかを判定するためのテストが行われる。適当なテストの1つは、((UnusedThreads) < (MinThreads) & (TotalThreads) < (MaxThreads))かどうかである。そうではない場合、ステップ654で管理スレッドが休眠状態に戻る。そうである場合、ステップ655で((UnusedThreads) - (MinThreads))個のスレッドをプールに加えるという式に応じて、通信スレッドがスレッド・プールに追加される。ステップ657では、(UnusedThreads)が(MinThreads)と等しくなるように設定される。ただし、UnusedThreadsが(MinThreads)限界より下がったときだけ、

例を示している。

【0148】

ただちにスレッドが追加されることに留意されたい。それ以外の場合は、次のタイマ間隔までスレッドが遅延される。ステップ659では、(TotalThreads)変数が増加したスレッドの数だけ増分される。管理スレッドはステップ654の休眠状態に戻る。

20

【0153】図19では、パフォーマンスを改善するために通信スレッド・プール内の未使用スレッドの数を低減するためのサーバ管理スレッド方法も示している。ステップ661では、通信スレッドの割振りまたは割振り解除のために定期的にスレッドを起こすタイマによって、スレッドが起こされる。ステップ663では、スレッド・プール内のスレッドが多すぎるかどうかを判定するためのテスト、たとえば、((ReservedThreads) < (MinThreads)) & ((UnusedThread) > (MinThreads))かどうかなどが行われる。そうではない場合、ステップ664で次の間隔までスレッドが休眠する。そうである場合、ステップ665でプール内のスレッドの数が1だけ低減される。ステップ667では、(TotalThreads)が1だけ減分される。所定の期間中にまったく活動が発生しない場合、結果的に(TotalThreads)変数が(MinThread)に戻ることになる。ステップ669では、スレッド・プール内の未使用スレッドが少なすぎるかどうかをテストが判定する。(ReservedThreads) > (TotalThreads) & ((ReservedThreads) < (MaxThreads))かどうかの式を使用することができる。そうである場合、((ReservedThreads) - (TotalThreads))を加えるという式に応じて、スレッドがスレッド・プールに追加される。ステップ673では、(TotalThreads)変数が(ReservedThreads)に設定される。管理スレッドはステップ674の休眠状態に戻る。

30

40

50

【0154】本発明の利点としては、セッションおよび通常クライアント要求の進入制御がある。さらに、クライアント要求に対応するためのスレッド資源がクライアント・タスクに認められた限界まで必ず保証されている。したがって、サーバ・スレッド資源を待ってクライアントをいつまでも停止させておくことはない。

【0155】すべてのクライアント間で共用可能な事前割振りスレッドの場合、クライアント要求に対応するスレッド実行経路でスレッドの作成または削除が行われないので、サーバのパフォーマンスによってクライアント要求への応答時間が改善される。非活動の期間中に割振り済みスレッドの数を（MinThreads）まで定期的にブルーニング・バックすることによって、オーバコミット・スレッド資源が最小限になる。プール内のサーバ・スレッドの総数は、構成値（MaxThreads）まで拡大することができる。非活動スレッドは最小限に維持されるので、これによりシステム・オーバヘッドが低減される。

【0156】（MaxThreads）と（MinThreads）は構成限界であり、システム管理者がアクセスできるようにすることができるので、本発明によって動的スレッド資源調整が達成される。これらは、手作業で構成して、導入システムに最適な最小値に調整することによって調整することができる。あるいは、クライアントと同時クライアント要求の数について、1日の様々な時間に統計をとり続け、これらの統計に基づいて自動的に（MinThreads）と（MaxThreads）の値を計算し調整することもできる。

【0157】マルチスレッド化クライアント／サーバ・

クラスTNetworkServer：公用TNetworkThreadHandle

{

公用：

TNetworkServer－クラス・コンストラクタ

～TNetworkServer－クラス・デストラクタ

AddClientPort－クライアント・ポート・プールにポートを追加する

RemoveClientPort－クライアント・ポート・プールからポートを除去す

る

AddServerThread－ThreadPoolにスレッドを追加する

DeleteServerThread－ThreadPoolからスレッドを削除する

ExecuteMgmtThread－管理スレッド入口

ExecuteThread－NetworkThread入口点

私用：

RegisterNames－クライアントにとって使用可能になるべきサーバ名を

公表する

【0161】

ネットワーク・サーバ実行用に使用するネットワーク・スレッド・クラス

クラスTNetworkThreadHandle：

{

公用：

TNetworkThreadHandle－クラス・コンストラクタ

TNetworkThreadHandle－クラス・デストラクタ

Fork－クラス関数を実行する新しいスレッドを開始する

Join－スレッドが完了するまで待つ

Release－スレッドがそのオブジェクトを削除できることを示す

【0162】

ネットワーク・クライアント／サーバ通信用のRPCクラス

class TNWRPCMessage {

公用：

システムではスレッド・プールの概念が一般に使用されてきたが、クライアントの数とクライアントの使用法に基づく通信スレッド・プールの動的調整はこれまで行われていない。本発明は、長期または短期で動作するクライアント要求に対応するために同様に適用可能である。さらに、ユーザ・レベルで動作するネットワーク・プロトコル・サーバを実現するためのスレッド・プールの使い方は、先行技術では実施されていない。

【0158】本発明は、サーバがマルチタスキング化され、多数のクライアントを効率よくサポートしなければならない、いかなるクライアント／サーバ・システムでも実施することができる。この解決策は、IBMメインフレームMVSや多くのUNIXベースのシステムなど、軽量の実行ユニットを持たないシステムには特に有用である。

【0159】前述のように、上記のアルゴリズムを使用するネットワーク・プロトコル・サーバは、1組のネットワーク・サーバ・クラスを定義することによって、オブジェクト指向技術を使用して実現することができる。

このようなクラスの定義例を以下に示す。

【0160】

```

43
virtual TNWRPCMessage();
virtual ~TNWRPCMessage();
/* クライアント側メソッド: SendRequest() */
virtual kern_return_t SendRequest (const port_t&
clientport,
    void* buffer, int& buflen): // buffer used for
inbound/outbound
    virtual kern_return_t SendRequest (const port_t&
sessionport,
    port_t& clientport, void* buffer, int& buflen):
/* サーバ側メソッド: ReceiveRequest() &
Reply&Receive() */
    virtual kern_return_t ReceiveRequest (const port_t&
sessionport
    void* buffer, int& buflen):
    virtual kern_return_t ReceiveRequest (const port_t&
clientport_pool,
    void* buffer, int& buflen, port_t& clientport):
    virtual kern_return_t SendReply (const port_t&
newclientport,
    void* buffer, int& buflen, port_t& sessionport):
    virtual kern_return_t SendReply (const port_t&
clientport,
    void* buffer, int& buflen):
};

```

44

【0163】クライアント／サーバ環境でのネットワーク要求の表現

ここでは、クライアント／サーバ・モデル用のネットワーク・プロトコル要求のオブジェクト指向表現を示す。プロトコルAPIがオブジェクト指向で、プロトコル・インプリメンテーションもオブジェクト指向の場合には、プロトコル要求のオブジェクト指向表現は重要なものになる。ネットワーク・プロトコルにアクセスするためのオブジェクト・ベースのクライアント要求がサーバに送られ、このサーバがこれらの要求を適切なネットワーク・プロトコル層オブジェクトに送達する。本発明は、クライアント・ネットワーク要求を転送し、その要求をサーバ上に常駐する適切なネットワーク・プロトコル層に送達し、要求の結果をクライアント・アプリケーションに取り出すための新しい方式を提示する。クライアント要求は「ネットワーク操作」オブジェクトで折り返されるが、そのオブジェクトはサーバがネットワーク・プロトコル層オブジェクトに要求を提示できるように必要な情報をすべて含んでいる。

【0164】以下のシナリオを検討されたい。クライアントAPIは1組のオブジェクト指向ネットワーク・プロトコル・インタフェース要求を含み、プロトコルはオブジェクトのスタックとして実現され、それぞれのオブジェクトは特定のOS I層を表している。ネットワーク・プロトコル・スタックは、ネットワーク・サーバ上に

常駐する様々なネットワーク・プロトコルを提供し、クライアントとサーバとの間でやりとりするためにRPCまたはIPCなどの通信機構が存在する。クライアントが何らかのデータの送信のようなネットワーク活動を要求した場合、サーバ上の適切なネットワーク・プロトコル・スタックにその要求を伝送する必要がある。本発明は、このようなクライアント要求をサーバに転送するための統一方式を提示する。この方式は多様性を利用しているので、このような要求を発送し、それをサーバで処理するプロセスは、どの要求およびプロトコル・スタックについても変わらない。

【0165】ネットワーク・プロトコルにアクセスするためのクライアント・インタフェースは、主にTProtocolInterfaceクラスによる。このプロトコル・インプリメンテーションは、TProtocolLayerクラスに基づいてそれぞれのOS I層を表している。このネットワーク・サブシステムは、TCP/IPやSNAなどの各プロトコル用のTFamilyLayerオブジェクトと、TDataLinkLayerオブジェクトから構成され、どちらのオブジェクトもシステム内に常駐している。セッション層、トランスポート層、ネットワーク層を表すためのTProtocolLayerオブジェクトのスタックはすべてのクライアント端点ごとに作成され、クライアント通信端点はTAccessDefinitionオブジェクトによって記述される。このような概念、関連クラス、その機能については、いずれも上記のプロトコ

30

40

50

ル・インタフェース・モデルの項で説明する。

【0166】1. ネットワーク操作オブジェクト
クライアントからサーバへのネットワーク要求とその逆のネットワーク要求は、いずれもTNetworkOperationオブジェクトを使用して転送される。TNetworkOperationオブジェクトは、サーバ内のプロトコル層にクライアント・ネットワーク要求を伝送し、その結果をサーバからクライアントにリレーするための機構を提供する。

【0167】TNetworkOperationクラスは、すべての要求の基本クラスである。クライアント・インタフェースがサーバに対して行う各要求ごとに、TNetworkProtocolからサブクラスが派生している。TProtocolInterfaceメソッドのそれぞれについて、対応する「操作オブジェクト」クラスが1つずつ定義されている。図20は、いくつかのクライアント要求用のクラス・オブジェクトのクラス階層を示している。TNetworkOperationクラス701は階層の一番上に位置する。TConnectOpクラス703と、TSendOpクラス705と、TReceiveOpクラス707と、TDisconnectOpクラス709と、TBindOpクラス713は、いずれも基本クラスから派生したもので、TProtocolInterfaceクラスの接続操作、送信操作、受信操作、切断操作、バインド操作にそれぞれ対応する。GetPeerNameおよびGetLocalNameなどの獲得および設定要求は、TGetSetNetworkOpクラス711という1つの操作クラスにバンドルされている。

【0168】TNetworkOperationオブジェクトは、ネットワーク・サーバでの対応を必要とする要求を満足するためにTProtocolInterfaceによって作成される。TNetworkOperationから派生したクラスは、インタフェースからの特定の要求を表す。TNetworkOperationオブジェクトは、要求が適切なプロトコル層に伝送されるように、RPC/IPC機構を使用してネットワーク・サーバに送られる。ネットワーク・サーバがNetworkOperationオブジェクトを受け取ると、それは操作オブジェクト上でExecute()メソッドを呼び出し、次に適切なプロトコル・インプリメンテーション層オブジェクト上で対応する関数を呼び出す。

【0169】したがって、TNetworkOperationオブジェクトは、それに対して要求したタスクを実行するために「組み込み知能」を備えている。ネットワーク・サーバは、TNetworkOperationオブジェクトを受け取ると、そのオブジェクト内のExecute()メソッドを呼び出す。クライアント要求の特徴とは無関係に、サーバ機能は同じである。各クライアント要求用のExecute()メソッドは、クライアント要求を適切なプロトコル層オブジェクトに伝送するために必要な情報をすべて含んでいる。

【0170】TNetworkOperationオブジェクトは、TProtocolInterfaceの具象クラスでも作成することができる。たとえば、TBindOpのデフォルト・クラスがTCP/IPの要求を満足しない場合、TCP/IPインタフ

ェースは、TTCPBindOpクラスを作成するようにTBindOpを指定変更する。その後、クライアントがバインド要求を行うと、具象TCP/IPクライアントAPIを表すTTCPINFクラスはTTCPBindOpオブジェクトを作成する。特定のクライアント要求の意味を再定義するか、または新しい要求とその操作オブジェクトを追加することをTNetworkOperationクラスから継承できる能力により、この方式は極めて柔軟かつ強力なものになる。

【0171】図21は、特定のクライアント要求を満たすために作成されるクラス・オブジェクトのクラス階層を示している。同図のTTCPBindOpオブジェクト715は、クライアント・アプリケーションからのバインド要求に対応するTTCPINF::Bind()メソッドによって作成される。これにより、TProtocolInterface::Bind()メソッドが指定変更される。TTCPNewOp717とTTCPNewOp719は、いくつかのクライアント要求の場合にTCP/IPに固有の2つの新しい操作オブジェクトの例である。

【0172】2. TNetworkOperationの関数

TNetworkOperationクラスが提供する重要な関数の一部を以下に示す。

【0173】Classコンストラクタ： この関数は、クライアント・アプリケーションによる要求の対象となるプロトコル層インデックスを設定するものである。

【0174】層インデックスは、その要求をプロトコル・スタックのトランスポート層、ネットワーク層、ファミリー層のいずれに送るべきかを識別するものである。

【0175】Execute()： この関数は、TNetworkOperationオブジェクトを受け取ったときにサーバによって呼び出される。この関数は、操作オブジェクトから層インデックスを獲得し、操作オブジェクトから関連パラメータを収集し、スタックの適切なTProtocolLayerオブジェクトへの呼出しを行うものである。たとえば、TBindOp::Execute()は、TTCPProtocolAddressオブジェクトをBind()関数へのパラメータとしてTProtocolLayer::Bind()を呼び出すはずである。

【0176】Get/SetLayerIndex： この関数は、操作オブジェクトで層インデックスを返す/設定するものである。

【0177】SetLocationToClient： デフォルトでは、操作オブジェクトがその位置をサーバに設定する。操作オブジェクトは、サーバ上にあるか、クライアント上にあるかによって、挙動が異なってくる。たとえば、TProtocolLayer::Bind()関数へのパラメータなので、TBindOpはTNetworkAddressオブジェクトをサーバに送る必要がある。しかし、サーバはアドレスをクライアントに戻す必要はない。位置フラグを使用して、パラメータの受渡しが制御される。ネットワーク操作オブジェクトがクライアントによって作成されると、そのオブジェクトは位置をクライアントに設定する。

【0178】Stream-out演算子： この関数は、操作オブジェクトを平坦化し、そのオブジェクトのデータ・メンバーをデータ・ストリームに入れるために使用する。たとえば、TBindOp用のStream-out演算子は、それがクライアントからのオブジェクトを送信する場合にTNetworkAddressオブジェクトを平坦化する。TSendOpは、バッファおよび宛先アドレスをサーバに送るためにバッファ・アドレスとTNetworkAddressオブジェクトを平坦化することができる。次にサーバはSendOp::Execute()を呼び出し、次にこれがユーザ・データとともにTProtocolLayer::Xmit()メソッドを呼び出して、宛先にデータを送る。送信が完了すると、サーバはRPC/IPC機構を使用してTSendCompletionオブジェクトをクライアントに戻す。TSendOp用のStream-out演算子は、それがサーバであるかどうかを検査し、TSendCompletionオブジェクトをストリームアウトする。

【0179】Stream-in演算子： この関数は、オブジェクトが平坦化された場合にデータ・ストリームからオブジェクトを再構築するために使用する。Stream-out演算子とは逆の操作であることは明らかである。操作オブジェクトは、データ・ストリームにオブジェクトを平坦化したり、データ・ストリームからオブジェクトを再構築するために位置フラグを使用する。

【0180】3. クライアントサーバ通信
ユーザが作成する通信端点についても、プロトコル層オブジェクトのスタックに端点を関連付けるために維持しなければならない固有のIDが存在していなければならない。プロトコル層オブジェクトは、サーバ・プロセス上の端点を表す。ただし、この対応は1対1であることに留意されたい。このため、TAccessDefinition::Instantiate()メソッドを使用して端点の作成中にTAccessOpオブジェクトが作成される。TAccessOpオブジェクトは、通信のクライアント側を表すClientStackHeadオブジェクトを作成する。次にTAccessOpオブジェクトは平坦化され、ClientStackHeadによりRPCまたはIPC機構を使用してサーバに送られる。次にサーバは、TNetworkOperationのstream-in演算子を使用してデータ・ストリームからTAccessOpオブジェクトを再構築し、TAccessOp::Execute()メソッドを呼び出す。この関数は、プロトコル・インタフェース・オブジェクトからプロトコル層オブジェクトを作成するServerStackHeadオブジェクトを作成し、TServerStackHeadオブジェクトでこれらのプロトコル層オブジェクトへのポインタのリストを保持する。TServerStackHeadポインタはネットワーク・サーバのグローバル・テーブルに格納され、インデックスはクライアントにストリームアウトされる。TClientStackHeadオブジェクトは、ServerStackHeadIDを格納し、後続のすべての操作にそれを使用する。したがって、ServerStackHeadIDは、クライアントとサーバとの間で固有のIDとして機能する。TBindOpなどの後続の要求

は、サーバによって受け取られると、TBindOpで渡されるIDを使用して、対応するサーバ・スタック・ヘッドの位置を特定する。

【0181】TClientStackHeadとTServerStackHeadは、所与の端点についてクライアントとサーバとのやりとりを管理する。これらのオブジェクトは、クライアント端点のハンドルであるTAccessDefinitionオブジェクトと、サーバ上の端点を表すTProtocolLayerオブジェクトの対応スタックとの間をリンクする。また、ClientStackHeadとServerStackHeadの対は、クライアントとサーバをリンクする内部管理オブジェクトを構成する。

【0182】TClientStackHeadの重要な関数の一部を以下に示す。

【0183】1. ProcessOperation： この関数は、TClientStackHeadがTNetworkOperationオブジェクトを処理するために、TProtocolInterfaceまたはTAccessDefinitionオブジェクトによって呼び出される。この関数は、TNetworkOperationオブジェクトを平坦化し、システムが提供するRPC/IPC機構を使用して、平坦化した操作オブジェクトをサーバに送る。また、この関数は、送られた要求に対してサーバが応答したときにNetworkOperationオブジェクトも再構築する。基本的にこの関数は、サーバとの間でNetworkOperationオブジェクトを送受信するものである。

【0184】2. SaveServerInfo： このメソッドは、ServerStackHeadIDをClientStackHeadに保管するために呼び出されるものである。AccessDefinitionがインスタンス化されると、サーバは端点用に作成されたServerStackHead用のIDとともにTAccessOpオブジェクトを返す。その後、サーバに対して要求が送られるときは、必ずNetworkOperationがこのIDを使用する。

【0185】3. CancelRequests： この関数は、クライアント・アプリケーションによってTProtocolInterface::CancelRequestsが呼び出されたとき、またはクライアント・アプリケーションが異常終了したときに呼び出されるものである。ClientStackHeadは、必要な終結処理を行うようServerStackHeadに通知するCancelRequestOp操作オブジェクトを作成する。

【0186】ServerStackHeadの重要な関数の一部を以下に示す。

【0187】1. Classコンストラクタ： このコンストラクタは、TProtocolInterfaceオブジェクトのスタックを受け取り、対応するTProtocolLayerオブジェクトを構築する。これは、後続の要求からこれらの層オブジェクトを呼び出すためにこれらの層オブジェクトへのポインタを維持している。

【0188】2. ProcessOperation： この関数は、クライアントからNetworkOperationオブジェクトを受け取ったときにNetworkServerProcessによって呼び出されるものである。この関数は、適切なProtocolLayerオブジ

エクトの位置を特定した後、TNetworkOperationオブジェクト上でExecute()メソッドを呼び出す。次にExecute()メソッドは、ProtocolLayerオブジェクト上で必要なメソッドを呼び出す。

【0189】3. GetProtocolLayer: この関数は、インデックスが付けられたTProtocolLayerオブジェクトへのポインタを返すものである。このインデックスはクライアント・アプリケーションによって渡される。

【0190】4. CancelRequests: この関数は、プロトコル・スタック上のすべての保留要求を取り消すためにTCancelRequestsOpを受け取ったときに呼び出されるものである。これは、TClientStackHeadのCancelRequestsに対応するサーバ側の関数である。

【0191】図22は、上記のオブジェクトのクラス階層を示し、Boochの表記法を使用してオブジェクト・モデルを記述している。TAccessDefinition101は、TNetworkDefinition100から派生したものである。TAccessDefinition101は、端点を構成する様々なTProtocolInterface135オブジェクトを含んでいる。また、TAccessDefinition101は、クライアントサーバ通信のプリミティブ機能のすべてを実行するTClientStackHead721も含んでいる。TServerStackHead723は、ClientStackHeadに対応するサーバ側のクラスである。TClientStackHeadとTServerStackHeadの対は、プロトコル・インタフェースとプロトコル・インプリメンテーション層オブジェクトとのリンクを表し、したがって、クライアント・アプリケーションとサーバ上のプロトコル・スタックとをリンクする。TProtocolInterfaceクラス135とTProtocolLayerクラス151はともに共通基本クラスであるMProtocolService133から派生したものである。クライアントからのネットワーク要求はいずれもTNetworkOperation701オブジェクトで折り返されるサーバに送られる。すなわち、TProtocolInterface内のメソッドが適切なTNetworkOperationオブジェクトを作成し、操作オブジェクトはTClientStackHeadによって平坦化され、サーバに送られる。TNetworkOperationオブジェクトは、TClientStackHeadとTServerStackHeadの両方を使用する。

【0192】図23は、クライアントからサーバへの要求とその逆の要求の流れを示している。前述のように、端点は、クライアント上のTAccessDefinitionオブジェクトと、サーバ上のTProtocolLayerオブジェクトのスタックによって表される。クライアントとサーバとのやりとりは、TClientStackHeadオブジェクトとTServerStackHeadオブジェクトによって管理される。同図は2つの端点725を示している。クライアントからのネットワーク要求は、システムのRPCまたはIPC機構を使用してTNetworkOperationオブジェクトとしてServerProcess/Thread727に送られる。ServerProcessは、TNetworkOperationオブジェクトを再構築し、次に端点1を表すT

ServerStackHeadオブジェクトの位置を特定し、ServerStackHead1729に要求を経路指定する。サーバ上のこの端点を表すServerStackHead2へ端点2から要求を経路指定する場合も同様の処理が行われる。次に、ServerStack1はTNetworkOperation::Execute()メソッドを呼び出し、これがスタック1733のTTransportLayerの適切なメソッドを呼び出す。TTransportLayerは、要求をTFamilyLayer737に送るTNetworkLayerのメソッドの呼出しを選ぶこともできる。次にファミリー層は、スタック1733のTDataLinkLayerに要求を送る。TTransportLayerは、要求をTFamilyLayer737に送るTNetworkLayerのメソッドの呼出しを選ぶこともできる。次にファミリー層は、要求の処理後、アダプタに要求を送ることができるTDataLinkLayer739に要求を送る。DataLinkLayerは、アダプタを介して何らかのパケットを受信すると、そのパケットをファミリー層にディスパッチする。TFamilyLayerは、そのパケットを適切なスタックに経路指定する。呼出しからTTransportLayerに戻ると、TNetworkOperation::Execute()は応答を収集し、システムのRPC/IPC機構を使用して適切なTClientStackHeadにTNetworkOperationオブジェクトを戻す。この手続きは、サーバ上の端点を表す端点2とスタック2735上のいかなる要求についても同じである。

【0193】図23は、クライアントとサーバのクラス・オブジェクト間の関係を示している。以下の例では、一般的なネットワークのクライアント/サーバ・モデルを想定している。

【0194】3. 例

TCP/IPトランスポート・インタフェースを表すTTCPINFについて検討する。TTCPINF::Bind()が入力パラメータとしてTTCPAddressを取り、TTCPAddressオブジェクトを返すものと想定する。ただし、TTCPAddressはTProtocolAddressから派生したものであることに留意されたい。図24に示すように、TTCPINF::Bind()メソッドは以下のように実行する。

【0195】ステップ801では、TNetworkOperationオブジェクトであるTTCPBindOpが作成される。

【0196】ステップ803では、このAccessDefinition用のClientStackHeadオブジェクトを獲得する。

【0197】ステップ805では、TTCPBindOpオブジェクトのTTCPAddressを平坦化し、要求をサーバに送るために、ClientStackHead::ProcessOperation()が呼び出される。サーバは、ステップ807でRPC/IPCなどのメッセージ・ストリームから平坦化したTTCPBindOpを再構築する。

【0198】ステップ809では、サーバは、TTCPBindOpオブジェクトで渡されたスタックIDからServerStackHeadオブジェクトの位置を特定する。

【0199】ステップ811では、ServerStackHeadオブジェクトが適切なプロトコル層オブジェクトにTProto

collayer::Bind()を呼び出す。

【0200】ステップ813では、バインドするための呼出しがTTCPAddressオブジェクトを返し、サーバがTTCPBindOpで復帰情報(アドレス)を復元し、それをクライアントにストリームで戻す。ClientStackHeadはメッセージ・ストリームからTTCPBindOp()オブジェクトを再構築する。最後に、ステップ817では、TTCPINF::Bind()がTTCPBindOpからTTCPAddressを取り出し、それをクライアント・プログラムに返す。

【0201】特定の実施例に関連して本発明を示し説明してきたが、当業者であれば、修正を加え他の環境で本発明を実施できることを理解できるだろう。たとえば、上記の本発明はソフトウェアによって選択的に再構成または活動化される汎用コンピュータで都合よく実施することができるが、当業者は、上記の発明を実行するために特別に設計された専用装置を含む、ハードウェア、ファームウェア、またはソフトウェアとファームウェアとハードウェアの組合せで本発明を実施できることに留意されたい。したがって、特許請求の範囲に記載する本発明の精神および範囲を逸脱せずに、形式および細部の変更を行うことができる。

【0202】まとめとして、本発明の構成に関して以下の事項を開示する。

【0203】(1)サーバ・システム内の実行ユニットの、クライアント・プロセスとサーバ・プロセスとの間の通信プロセス専用のプールを動的に管理するための方法において、通信プロセス・プール内に、典型的なクライアント・ロードをサポートするのに必要な数である最小数、およびサーバ・システムにとって過負荷にならずにピーク・クライアント・ロードをサポートするための上限である最大数の実行ユニットを割り振るステップと、クライアントによるサービス要求をサーバ・システムが受信するステップとを含み、受信した各クライアント要求ごとに、受信したクライアント要求に実行ユニットを割り当てると、通信プロセス・プール内の現行数の実行ユニットが最大数の実行ユニットを上回ることになるかどうかを判定し、上回る場合にクライアント要求を拒否するステップと、受信したクライアント要求に実行ユニットを割り当てると、その要求を行うクライアント・タスクに割り当てられた現行数の実行ユニットがクライアント・タスク用の割り当てられた実行ユニットの数を上回ることになるかどうかを判定し、上回る場合にクライアント要求を拒否するステップと、判定ステップが否定であればクライアント要求を許可し、その結果、通信プロセス・プール内の実行ユニットがクライアント要求に割り当てられるステップとを含むことを特徴とする方法。

(2)サーバ・システム内の実行ユニットの、クライアント・プロセスとサーバ・プロセスとの間の通信プロセス専用のプールを動的に管理するためのシステムにおい

て、通信プロセス・プール内に、典型的なクライアント・ロードをサポートするのに必要な数である最小数、およびサーバ・システムにとって過負荷にならずにピーク・クライアント・ロードをサポートするための上限である最大数の実行ユニットを割り振る手段と、クライアントによるサービス要求をサーバ・システムが受信する手段と、受信したクライアント要求に実行ユニットを割り当てると、通信プロセス・プール内の現行数の実行ユニットが最大数の実行ユニットを上回ることになるかどうかを判定する手段と、受信したクライアント要求に実行ユニットを割り当てると、その要求を行うクライアント・タスクに割り当てられた現行数の実行ユニットがクライアント・タスク用の割り当てられた実行ユニットの数を上回ることになるかどうかを判定する手段と、通信プロセス・プール内の実行ユニットをクライアント要求に割り当てることができるという判定が確認した場合にクライアント要求を許可する手段とを含むことを特徴とするシステム。

(3)サーバ・システム内の実行ユニットの、クライアント・プロセスとサーバ・プロセスとの間の通信プロセス専用のプールを動的に管理するためのコンピュータが読取り可能な媒体上のコンピュータ・プログラム製品において、通信プロセス・プール内に、典型的なクライアント・ロードをサポートするのに必要な数である最小数、およびサーバ・システムにとって過負荷にならずにピーク・クライアント・ロードをサポートするための上限である最大数の実行ユニットを割り振るようにシステムに指示する手段と、クライアントによるサービス要求をサーバ・システムが受信するようにシステムに指示する手段と、受信したクライアント要求に実行ユニットを割り当てると、通信プロセス・プール内の現行数の実行ユニットが最大数の実行ユニットを上回ることになるかどうかを判定するようにシステムに指示する手段と、受信したクライアント要求に実行ユニットを割り当てると、その要求を行うクライアント・タスクに割り当てられた現行数の実行ユニットがクライアント・タスク用の割り当てられた実行ユニットの数を上回ることになるかどうかを判定するようにシステムに指示する手段と、通信プロセス・プール内の実行ユニットをクライアント要求に割り当てることができるという判定に回答してクライアント要求を許可するようにシステムに指示する手段とを含むことを特徴とするコンピュータ・プログラム製品。

【図面の簡単な説明】

【図1】本発明の教示により構成されたコンピュータ・システムを示す図である。

【図2】ネットワーク定義オブジェクト用のクラス階層を示す図である。

【図3】ネットワーク・アドレス・クラス用のクラス階層を示す図である。

【図4】プロトコル・インタフェース・クラス用のクラス階層を示す図である。

【図5】プロトコル層クラス用のクラス階層を示す図である。

【図6】接続指向遷移の状態図である。

【図7】無接続状態遷移の状態図である。

【図8】本発明のTCP/IP実施例のクラス関係を示す図である。

【図9】本発明によりネットワーク接続をセットアップするためのプロセスの流れ図である。

【図10】図9に示すネットワーク接続プロセスの専用ネットワーク層にある様々なオブジェクトの体系図である。

【図11】ネットワーク事象オブジェクト用のクラス階層を示す図である。

【図12】単一の通信端点から事象を収集するためのプロセスの流れ図である。

【図13】複数の通信端点から事象を収集するためのプロセスの流れ図である。

【図14】ネットワーク・プロトコル・サーバのクライアント要求を処理するために通信スレッドのプールを管理するための第1および第2の実施例の体系図である。

【図15】ネットワーク・プロトコル・サーバのクライアント要求を処理するために通信スレッドのプールを管理するための第1および第2の実施例の体系図である。

【図16】通信スレッドのプール用の管理プロセスの流れ図である。

【図17】通信スレッドのプール用の管理プロセスの流れ図である。

【図18】通信スレッドのプール用の管理プロセスの流れ図である。

【図19】通信スレッドのプール用の管理プロセスの流れ図である。

【図20】ネットワーク操作オブジェクト用のクラス階層図である。

【図21】ネットワーク操作オブジェクト用のクラス階層図である。

【図22】本発明の様々なクラス間のクラス関係を示す図である。

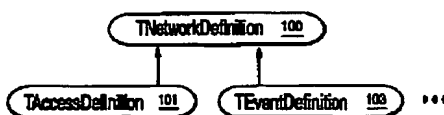
【図23】本発明によるネットワーク内の様々なオブジェクト間のメッセージの流れを示す図である。

【図24】ネットワーク操作オブジェクトでネットワーク・プロトコル要求を渡すための流れ図である。

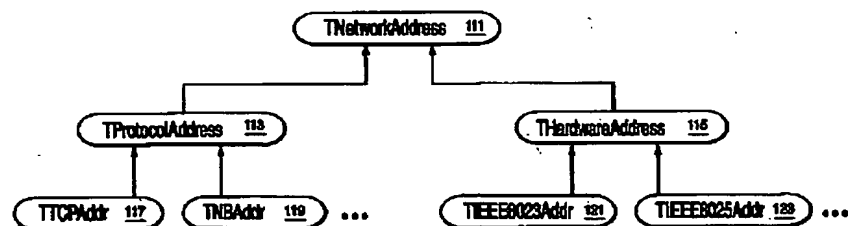
【符号の説明】

- 10 11 システム・ユニット
12 キーボード
13 マウス
14 グラフィック・ディスプレイ
15A スピーカ
15B スピーカ
21 システム・バス
22 マイクロプロセッサ
23 ROM
24 RAM
25 メモリ管理チップ
26 ハード・ディスク・ドライブ
27 フロッピー・ディスク・ドライブ
28 キーボード制御装置
29 マウス制御装置
30 ビデオ制御装置
31 オーディオ制御装置
32 CD-ROM
33 デジタル信号プロセッサ
40 入出力制御装置
30 46 ネットワーク
48 オペレーティング・システム
49 クライアント・アプリケーション
50 クライアント・インタフェース
51 プロトコル・オブジェクト
52 サーバ・アプリケーション

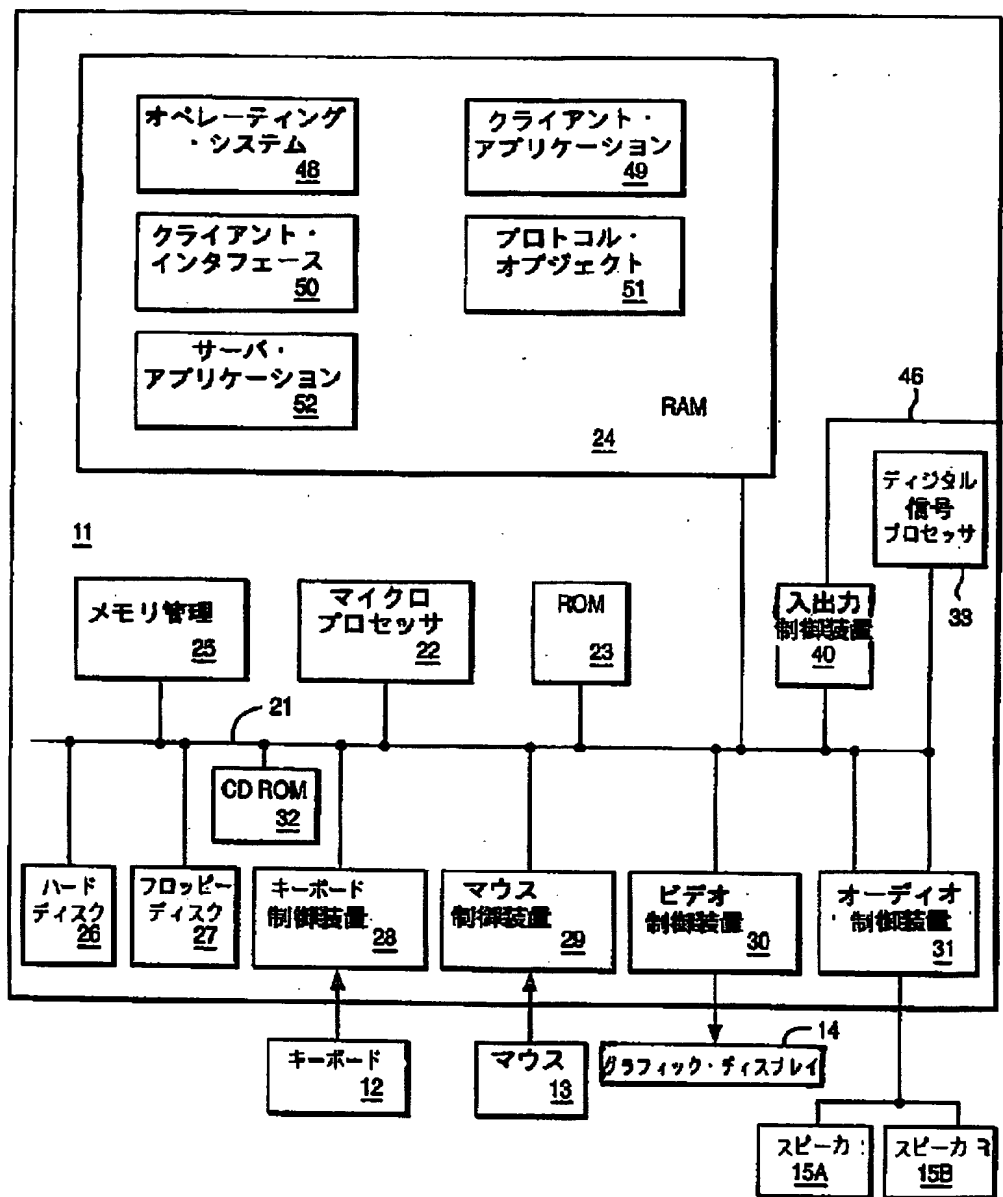
【図2】



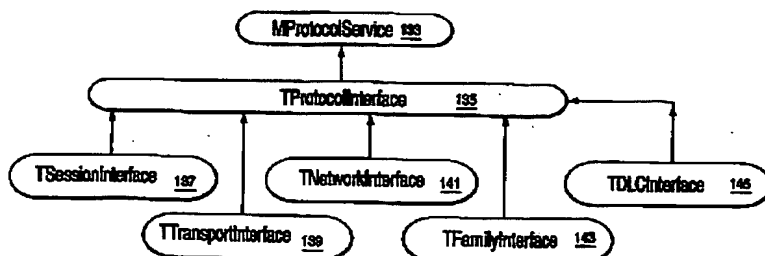
【図3】



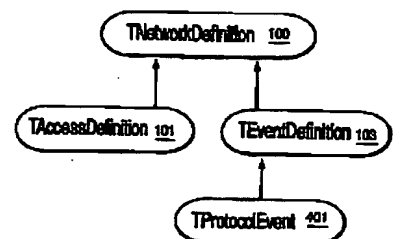
【図 1】



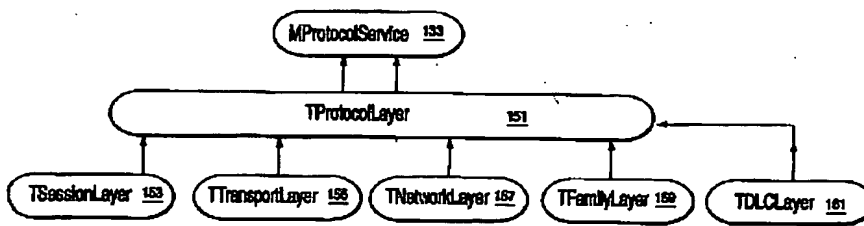
【図 4】



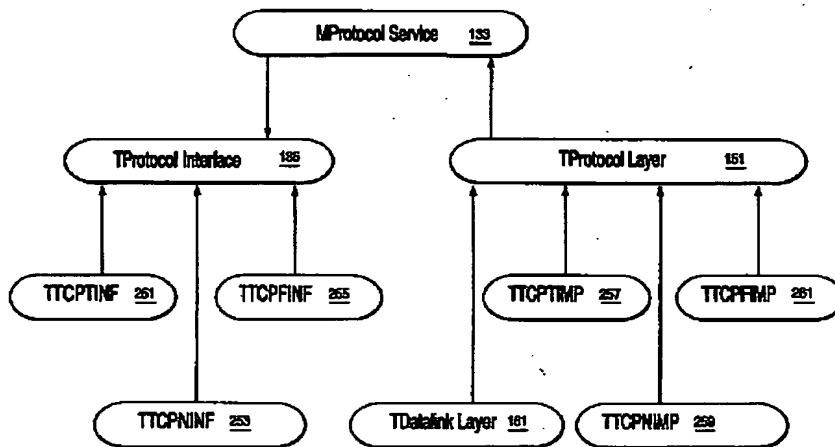
【図 11】



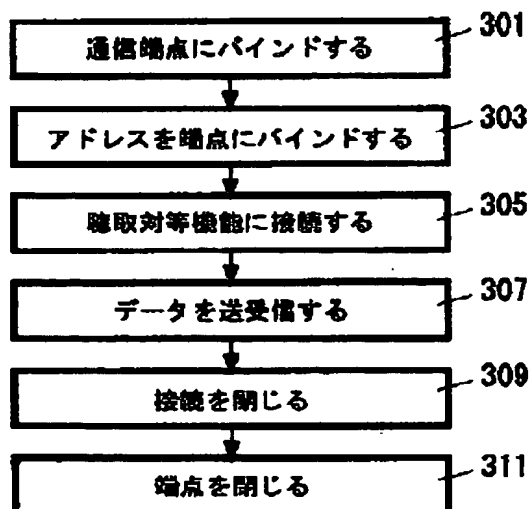
【図 5】



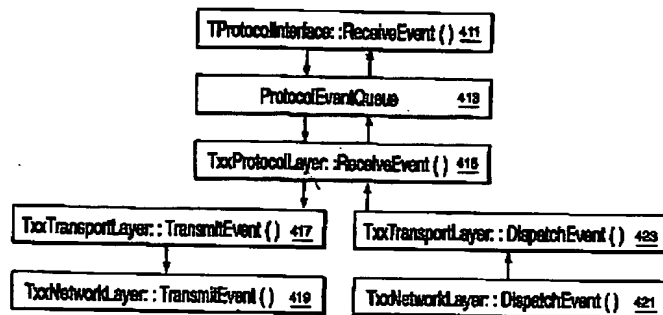
【図 8】



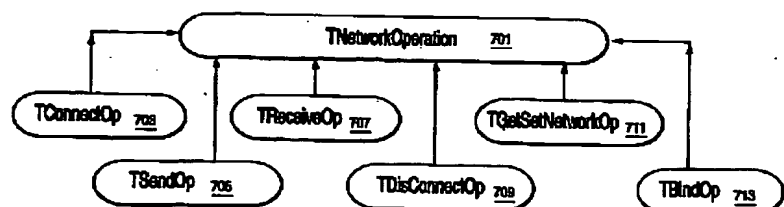
【図 9】



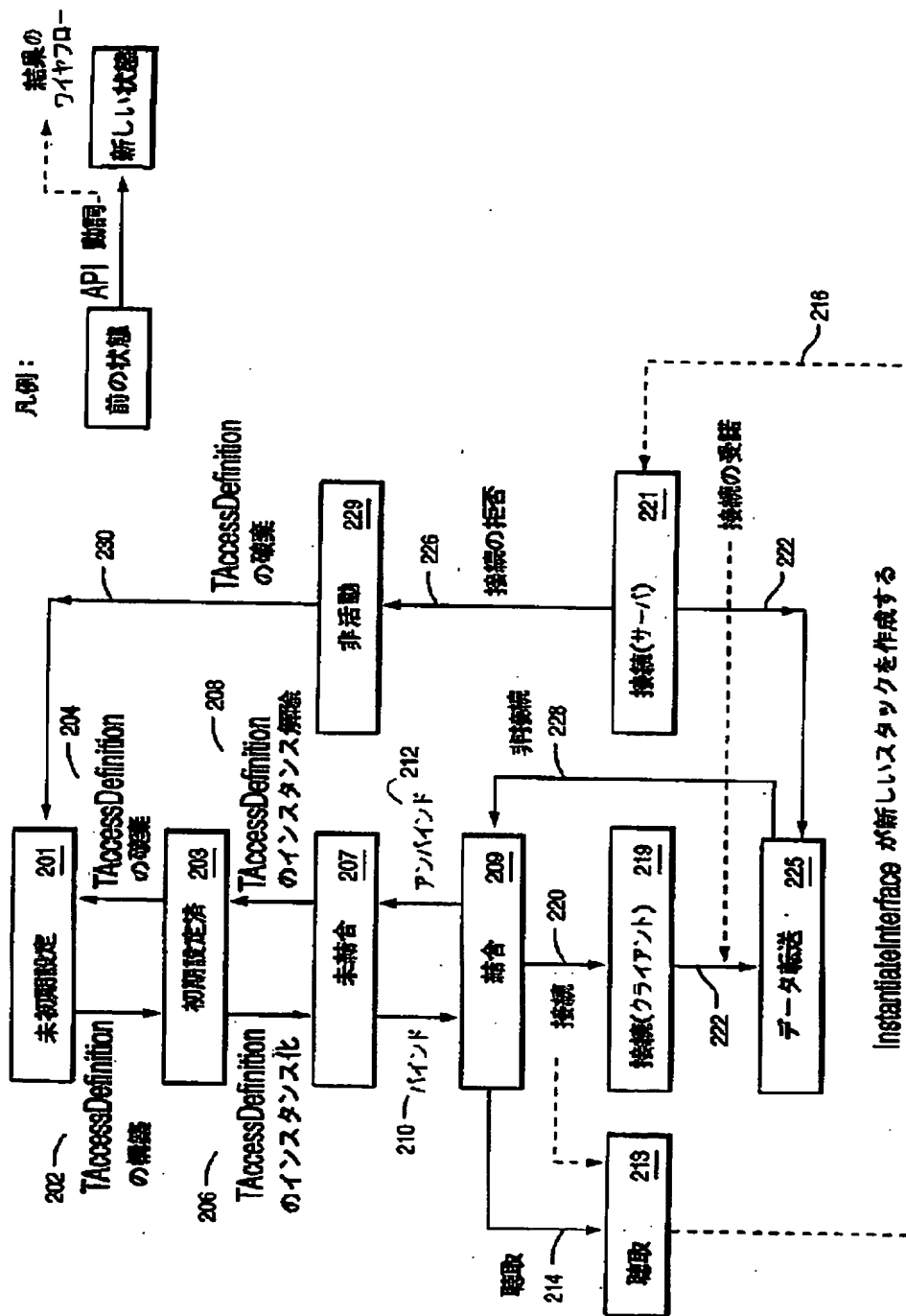
【図 12】



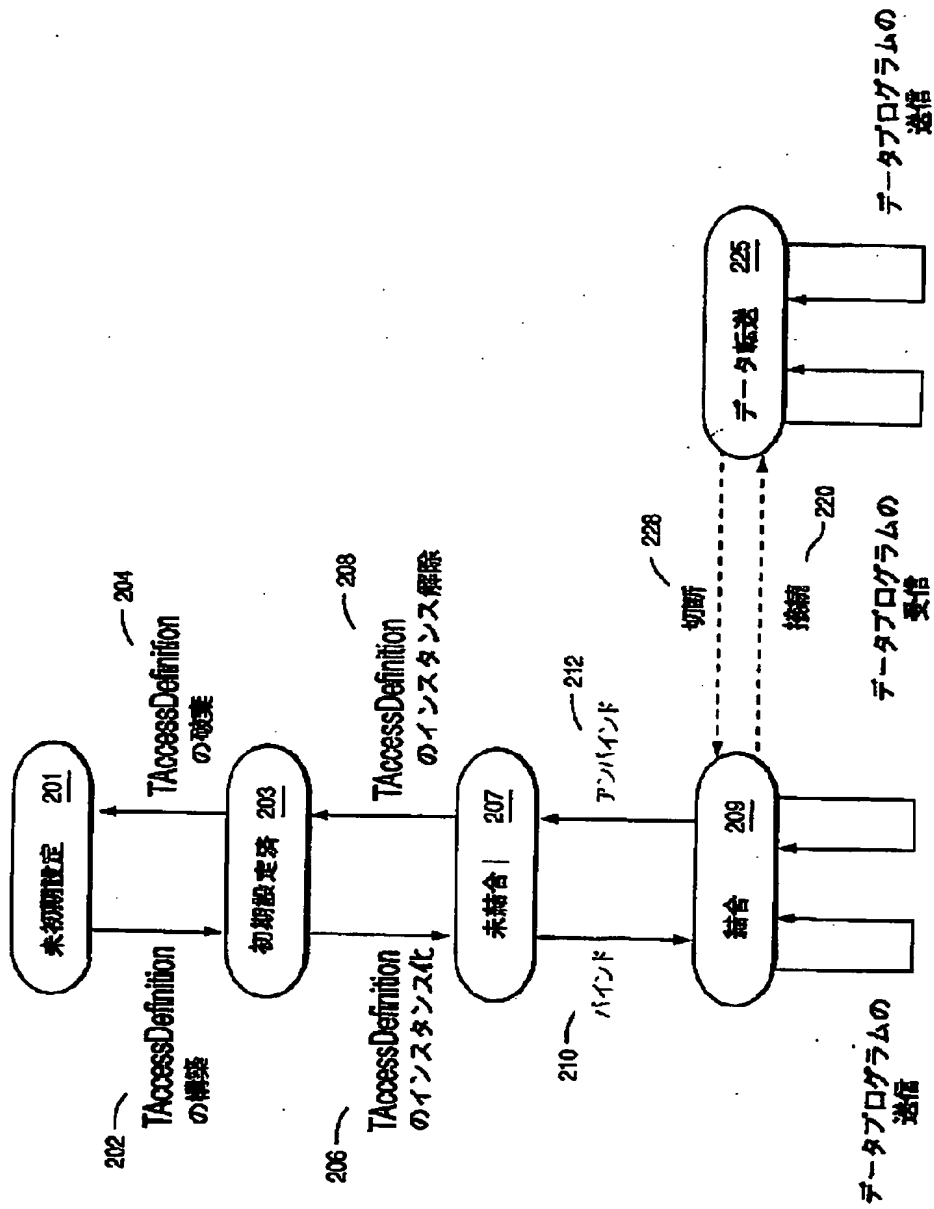
【図 20】



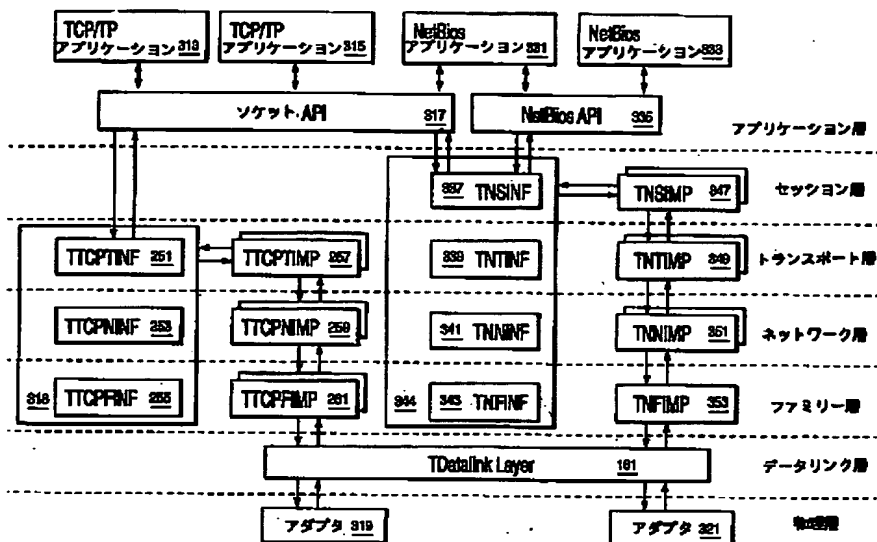
【図 6】



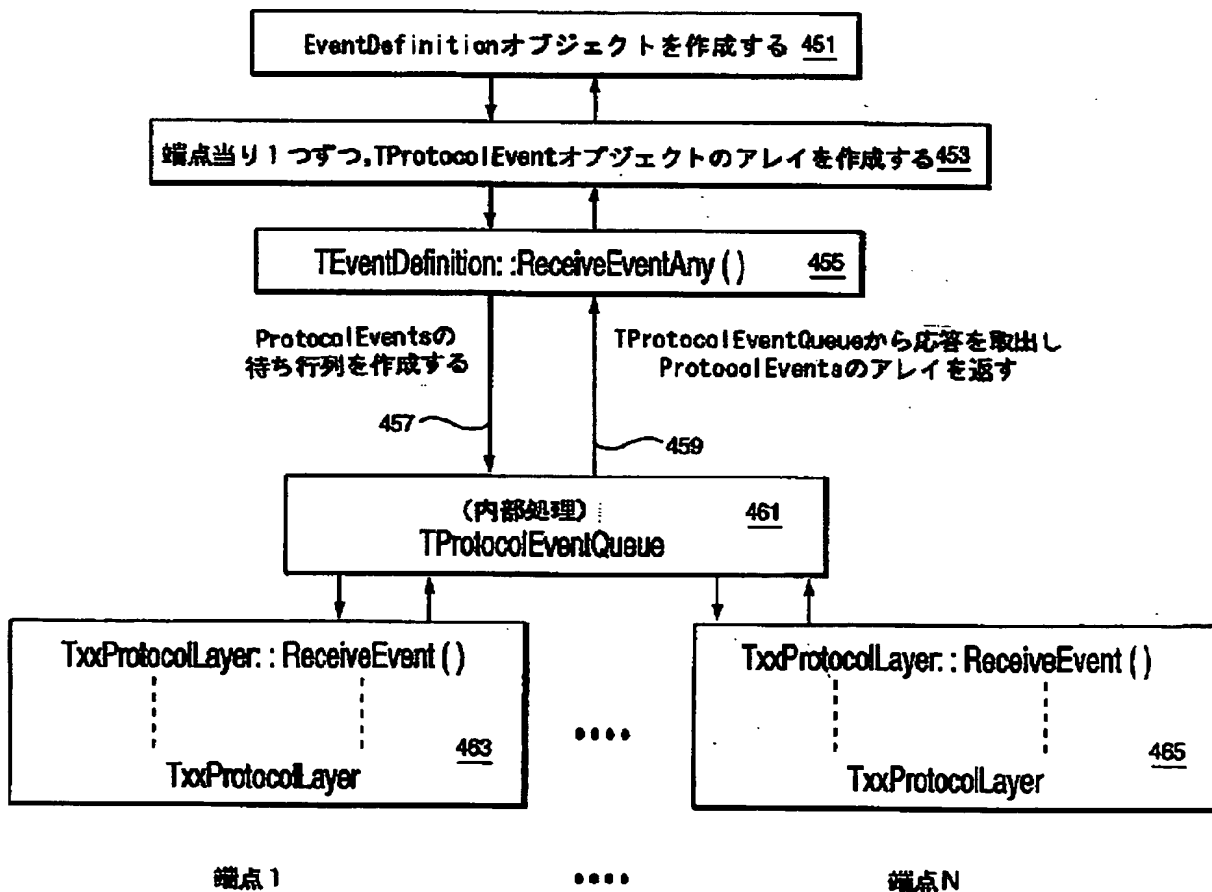
【図 7】



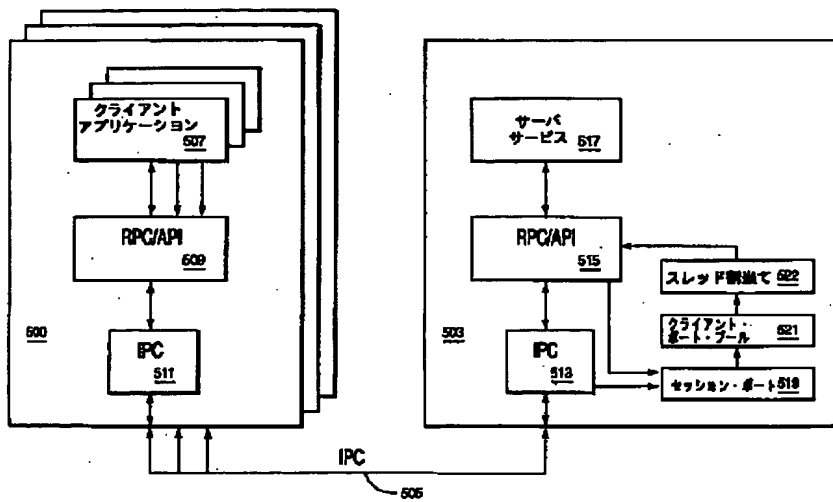
【図10】



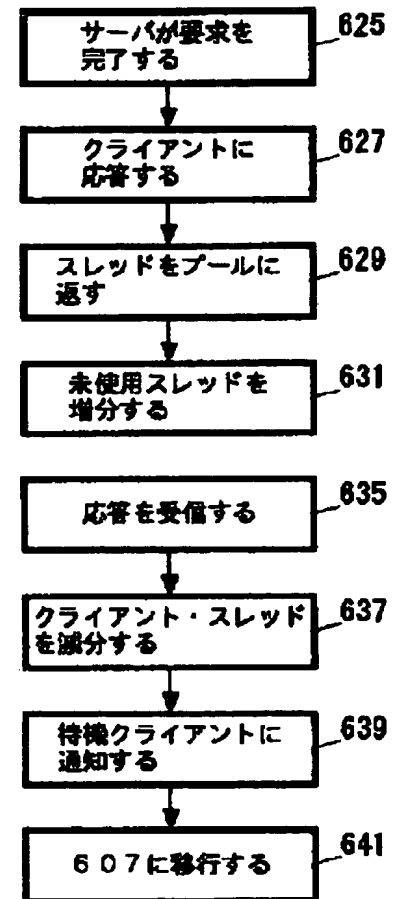
【図13】



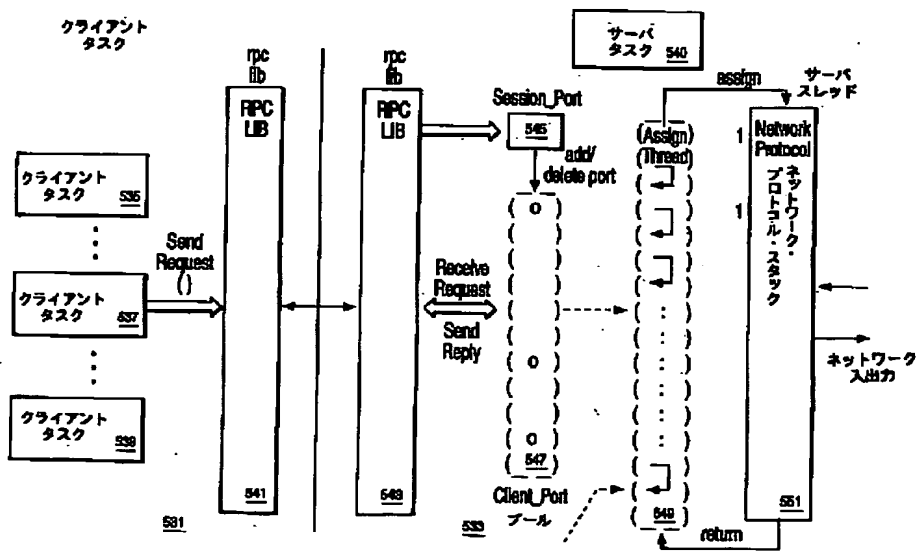
【図14】



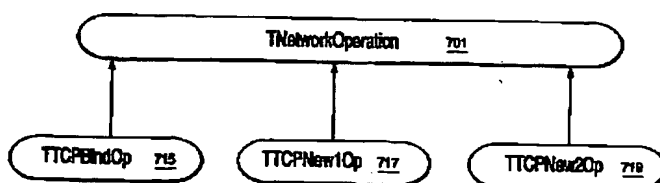
【図18】



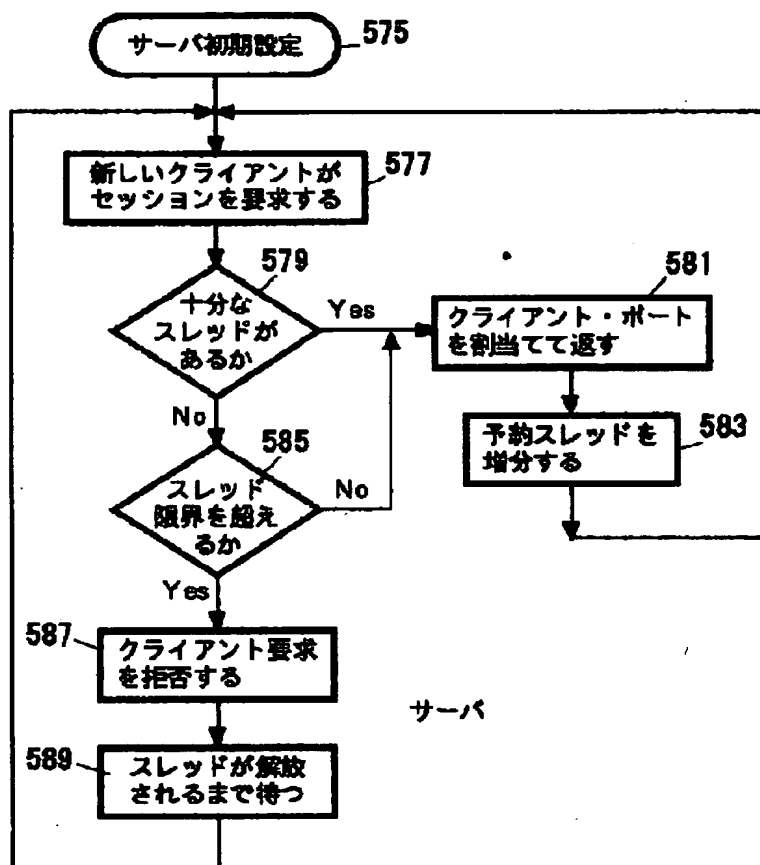
【図15】



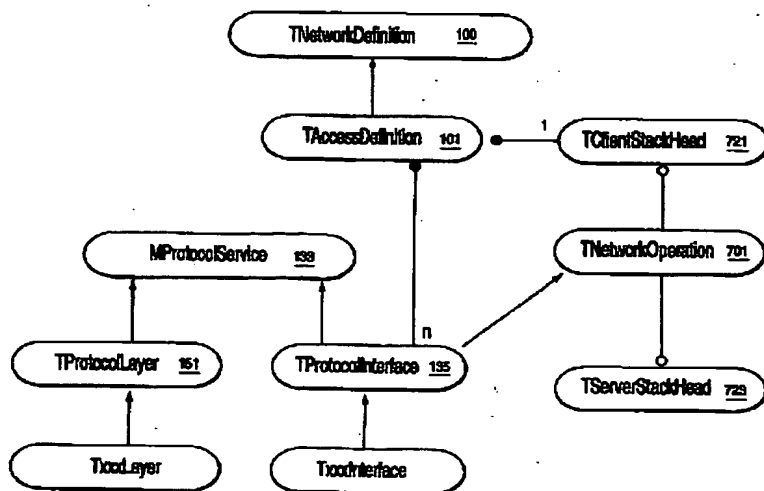
【図21】



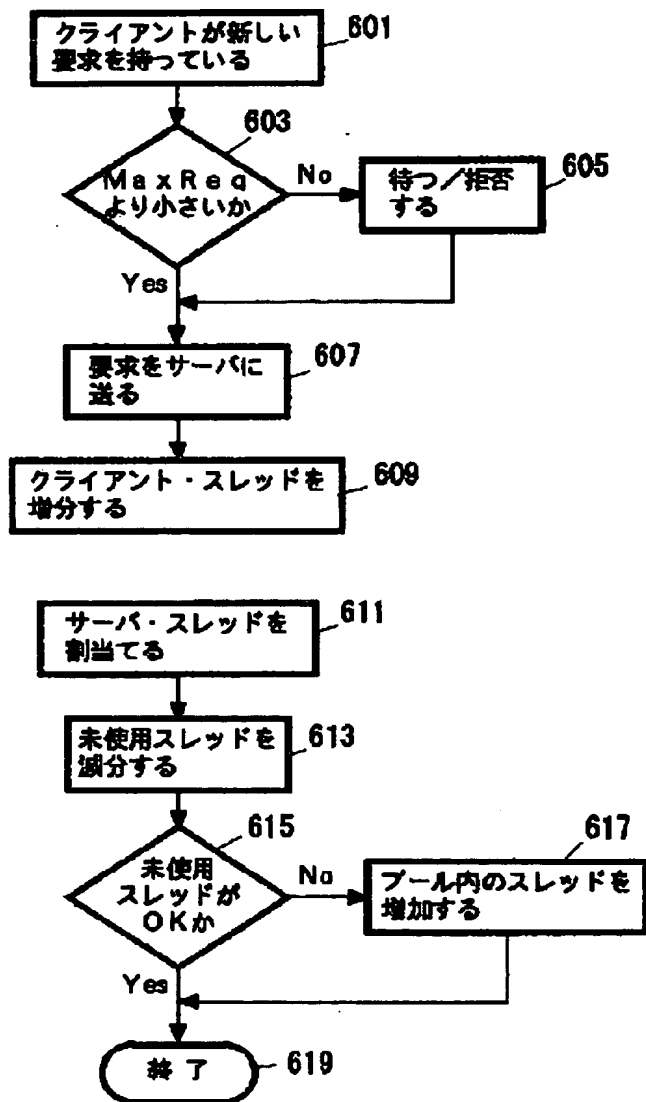
【図16】



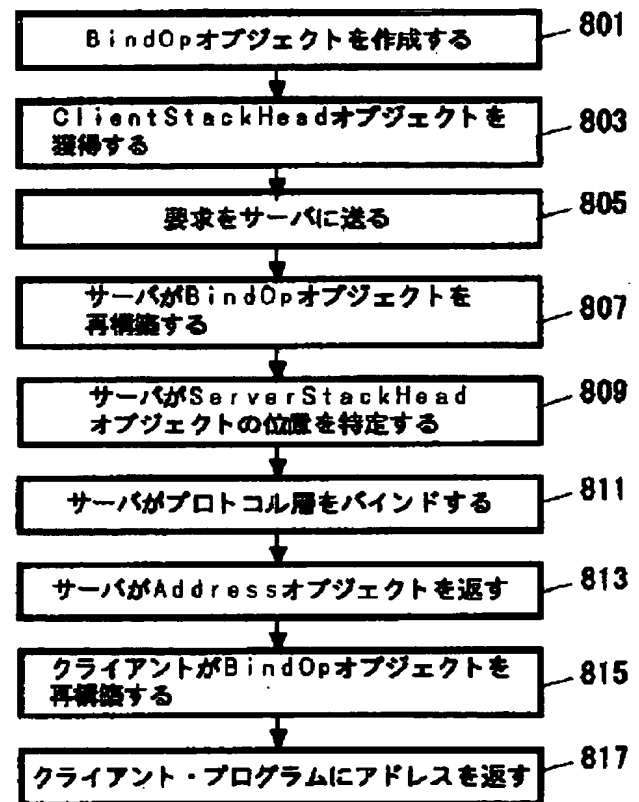
【図22】



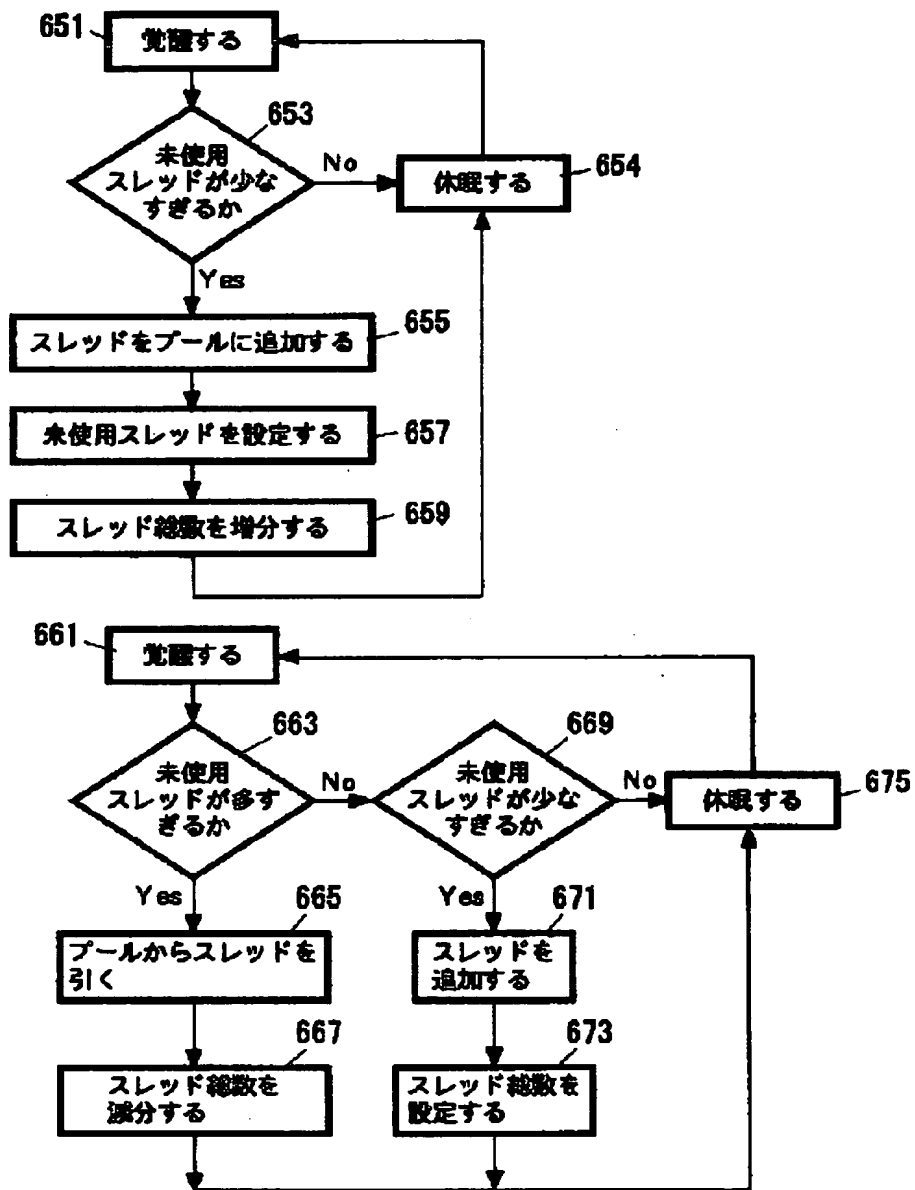
【図17】



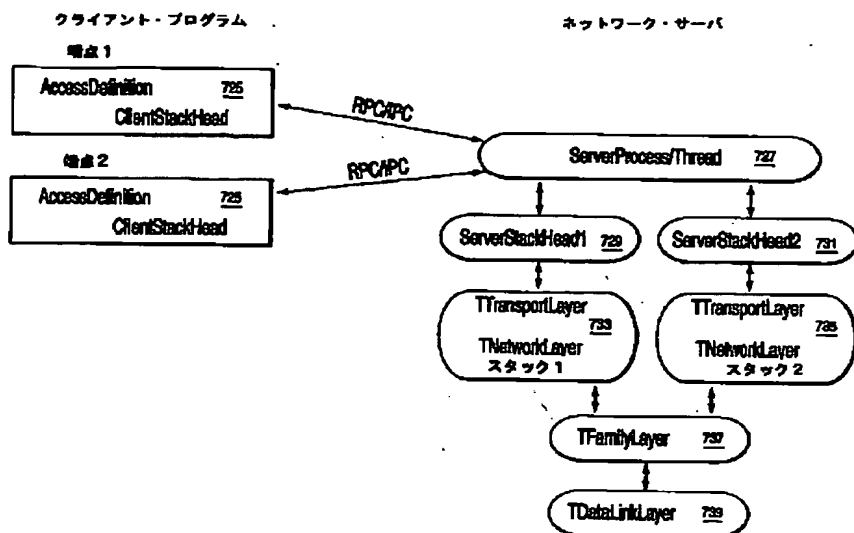
【図24】



【図19】



【図23】



フロントページの続き

(72)発明者 レオ・ユエ・タク・ユング
 アメリカ合衆国78759 テキサス州オース
 チン ディーケイ・ランチャ・ロード
 11714